A Language-centered Approach to Support Environmental Modeling with Cellular Automata

D i s s e r t a t i o n

zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)
im Fach Informatik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät II
der Humboldt-Universität zu Berlin

von
Diplom-Geograph Falko Martin Theisselmann

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Dr. Doris Dransch
2. Prof. Dr. Joachim Fischer
3. Prof. Dr. Michael Sonnenschein

Tag der Verteidigung: 19.11.2013

Abstract

The application of methods and technologies of software engineering to environmental modeling and simulation (EMS) is common, since both areas share basic issues of software development and digital simulation. Recent developments within the context of "Model-driven Engineering" (MDE) aim at supporting the development of software systems at the base of relatively abstract models as opposed to programming language code. A basic ingredient of MDE is the development of methods that allow the efficient development of "domain-specific languages" (DSL), in particular at the base of language metamodels. DSLs are mainly characterized by providing expressive means for the specification of models that are particularly tailored towards the needs of users.

Besides functional properties, a distinguishing feature of DSLs is the provision of non-functional properties that are related to pragmatics. However, existing investigations of MDE in software engineering and particularly EMS primarily focus on technical considerations, e.g. feasibility. This thesis shows how MDE and language metamodeling in particular, may support pragmatic aspects that reflect epistemic and cognitive aspects of scientific investigations. For this, DSLs and language metamodeling in particular are set into the context of "model-based science" and "model-based reasoning". It is shown that the specific properties of metamodel-based DSLs may be used to support those properties, in particular transparency, which are of particular relevance against the background of uncertainty, that is a characterizing property of EMS.

The findings are the base for the formulation of an corresponding specific metamodel-based approach for the provision of modeling tools for EMS (Language-centered Approach, LCA). The implementation of a corresponding exemplary modeling tool provides a DSL (ECAL) for modeling with Cellular Automata that has been developed within this thesis. ECAL (Environmental Cellular Automata Language) is particularly tailored to reflect the formerly discussed pragmatic aspects of Cellular Automata modeling in EMS. The concepts of ECAL follow from a characterization of relevant Cellular Automata models and corresponding modeling tools as described in literature. ECAL represents a level of abstraction that aims at being general with respect to typical degrees of freedom of Cellular Automata for EMS, but it does not provide typical degrees of freedom of typical modeling tools, which follows from above mentioned considerations of pragmatics. In order to proof applicability of LCA and ECAL, ECAL has been used to reimplement published models.

A particularly important feature of EMS is the inclusion of geo-spatial data (geodata). The integration of technology for processing geodata and simulation technology is a well-recognized field of scientific investigation. However, the integration of corresponding technologies at the base of metamodels is widely unrecognized, although influential efforts of standardization of geodata are widely based on language metamodels. This thesis suggests the integration of geodata processing and corresponding software at the base of metamodels. The modeling tool developed in this thesis shows the practical applicability of concepts.

Zusammenfassung

Die Anwendung von Methodiken und Technologien aus dem Bereich der Softwaretechnik auf den Bereich der Umweltmodellierung ist eine gemeinhin akzeptierte Vorgehensweise, da diese Bereiche grundsätzliche Problemstellungen bzgl. der Softwareentwicklung und der digitalen Simualation teilen. Im Rahmen der "modellgetriebenen Entwicklung"(MDE, model-driven engineering) werden Technologien entwickelt, die darauf abzielen, Softwaresysteme vorwiegend auf Basis von im Vergleich zu Programmquelltexten relativ abstrakten Modellen zu entwickeln. Ein wesentlicher Bestandteil von MDE sind Techniken zur effizienten Entwicklung von "domänenspezifischen Sprachen"(DSL, domain-specific language), die auf Sprachmetamodellen beruhen.

DSLs sollen eine besonders den Bedürfnissen der Nutzer angepasste Auswahl an Ausdrucksmitteln zur Verfügung stellen. Neben den funktionalen und technischen Eigenschaften sind es besonders die nicht-funktionalen Eigenschaften der Pragmatik, welche die Besonderheit der domänenspezifischen Sprachen ausmachen. Bestehende Untersuchungen von DSLs auf Basis von Technologien aus MDE im Bereich der Systementwicklung und insbesondere der Umweltmodellierung stellen v.a. technische Aspekte, z.B. Machbarkeitsüberlegungen, in den Vordergrund. Die vorliegende Arbeit zeigt, wie modellgetriebene Entwicklung, und insbesondere die metamodellbasierte Beschreibung von DSLs, darüber hinaus Aspekte der Pragmatik unterstützen kann, deren Relevanz im erkenntnistheoretischen und kognitiven Hintergrund wissenschaftlichen Forschens begründet wird. Hierzu wird vor dem Hintergund der Erkenntnisse des "modellbasierten Forschens"(model-based science und model-based reasoning) gezeigt, wie insbesondere durch Metamodelle beschriebene DSLs Möglichkeiten bieten, entsprechende pragmatische Aspekte besonders zu berücksichtigen, indem sie als Werkzeug zur Erkenntnisgewinnung aufgefasst werden. Dies ist v.a. im Kontext großer Unsicherheiten, wie sie für weite Teile der Umweltmodellierung charakterisierend sind, von grundsätzlicher Bedeutung.

Die Formulierung eines sprachzentrierten Ansatzes (LCA, language-centered approach) für die Werkzeugunterstützung konkretisiert die genannten Aspekte und bildet die Basis für eine beispielhafte Implementierung eines Werkzeuges mit einer DSL für die Beschreibung von Zellulären Automaten (ZA) für die Umweltmodellierung. Diese Sprache (Environmental Cellular Automata Language, ECAL) basiert auf einer Charakterisierung von ZA unter besonderer Berücksichtigung pragmatischer Aspekte, die aus einer Betrachtung relevanter ZA und entsprechender Modellierungswerkzeuge folgt. Es wird gezeigt, dass ECAL eine Abstraktionstufe darstellt, die allgemein bezüglich typischer Freiheitsgrade in der Umweltmodellierung ist, jedoch bewusst - und in pragmatischen Aspekten begründet - von vielen Möglichkeiten typischer Werkzeuge, absieht. Anwendungsfälle belegen die Verwendbarkeit von ECAL und der entsprechenden metamodellbasierten Werkzeugimplementierung.

Eine Besonderheit der Umweltmodellierung ist die Verwendung geo-räumlicher Daten (Geodaten). Die Kombination von Technologien zur Verarbeitung von Geodaten und Simulationstechnik ist ein wohlbekanntes Forschungsfeld, jedoch weitgehend unbeschrieben bzgl. metamodell-basierter Sprachen, obwohl einflussreiche Normen und Standards weitgehend auf demselben Metamodell-Formalismus beruhen. Die vorliegende Arbeit schlägt die metamodellbasierte Integration von GIS und Simulationstechnik vor und zeigt auf Basis des im Rahmen dieser Arbeit entwickelten Werkzeuges dessen praktische Anwendbarkeit.

Acknowledgements

When choosing an interdisciplinary approach to the investigations of this thesis, it was clear from the beginning that this enterprise will not succeed as an isolated effort. Insofar I want to express gratitude to those numerous people that supported me in various ways: First, I want to thank Prof. Dr. Joachim Fischer and Prof. Dr. Doris Dransch for giving me the opportunity to be part of the exciting project METRIK, but also for giving critical and constructive advice while facing my ideas with great openness.

Further, I want to thank the numerous colleagues within METRIK, GeoForschungsZentrum Potsdam and the Systems Analysis Group at Humboldt-University zu Berlin, who accompanied my work with various ways of support and constructive criticism. In particular, I want to thank Dr. Markus Scheidgen, Frank Kühnlenz and Prof. Dr. Tobia Lakes for their concrete inputs and fruitful collaboration. I want to particularly thank Dr. Daniel Sadilek and Dr. Andreas Kunert for the numerous inspiring discussions.

Last, but not least, I want to express gratitude for my family, first of all my wife and son, for taking all the inconveniences with a smile.

Contents

1 Introduction

1.1 Motivation

Environmental Modeling and Simulation (EMS) refers to the application of the method of modeling and simulation (M&S) in field of environmental science and management. EMS is a fundamental method for knowledge discovery and application is these fields. The growing availability of environmental data, interdisciplinary modeling and the need to apply scientific digital simulation models outside the scientific context define present requirements for EMS. In practice, model evolution, collaborative interdisciplinary modeling of complex dynamical systems and the use of scientific models in different technological environments is intellectually and technically challenging, in particular when we take limited resources w.r.t. to man-power, time and computing facilities into account.

Tools for M&S and EMS support modelers to face these challenges, where modelers are usually domain experts with a varying degree of education in respective technologies. Typical modeling tools are built upon basic system-theoretic assumptions and the fundamental theory of mathematical modeling and simulation (see Chapters 2.2.1 and 2.2.2). Moreover, the recent development of numerous tools for M&S and EMS with particular consideration of aspects of software engineering is closely related the development of a considerable body of knowledge concerning technical aspects of collaborative modeling and the reusability of models (see Chapters 3.3 and 3.4). In science however, the first goal of modeling is to gain insight into the structure and internal workings of systems, thus, the discovery of knowledge. Moreover, the feasibility of the application of gathered knowledge in environmental management requires credibility. It is widely accepted that sound scientific processes and credible application of knowledge outside science requires the assertion of "transparency" of model specifications, besides the consideration of well-known mathematical and technical issues. However, the notion of "transparency" is vague and there are conflicting perceptions of the role of models and their representation (see Chapter 2.4.4).

Domain-specific modeling languages (DSLs) are regarded to be particularly useful to address the issue of transparency. Therefore, a variety of DSLs for EMS has been developed, but the respective tools have limitations, in particular with respect to technical aspects of collaborative modeling and model reuse, where the costs of the implementation of tools that provide DSLs often appears to be the prohibiting factor. At a first sight, the concept of model-driven engineering (MDE) and the corresponding technologies seem to provide a feasible approach to address the current issue of collaborative modeling and the reuse of DSL-based models for mainly two reasons: First, MDE is based on the idea that models of systems are specified at a relatively abstract level by means of a number of related DSLs - in contrast to general-purpose programming languages. Second, MDE aims at the provision of technologies that make the implementation of DSL-providing modeling tools relatively efficient - in contrast to traditional tool and language implementation approaches.

However, the combination of the usage of DSLs for collaborative modeling that may have

the potential to combine transparency with powerful existing computational possibilities is a current issue in research of M&S and EMS (see Chapters 3.3.5 and 3.4.5). The few - mostly prototypical - developments primarily focus on technical aspects, but they show and discuss technical possibilities and limitations. However, a defining feature of DSLs is that these are assumed to incorporate exactly what modelers need, beyond purely functional and technical considerations. In the theory of computer languages these aspects are considered under the notion of "pragmatics". A characterization of pragmatic aspects has been developed in the field of programming languages and is under development for DSLs. However respective guidelines are typically conflicting, general and, if not general, rather specific for the context of software engineering, not for EMS (see Chapters 3.2.3 and 3.5.3). Against this background it appears that there is a lack of clarity concerning the application of DSLs in general and the application of MDE to EMS in particular. It is a goal of this thesis to contribute to the clarification of pragmatic aspects with particular consideration of aspects of transparency.

Cellular Automata is a modeling paradigm that exemplifies current pragmatic, thus philosophical, cognitive and technical issues in EMS in that it is typically applied in modeling systems where transparency is of particular importance. Although CA models are widely used in environmental science and there exist numerous tools that provide DSLs for CA modeling, CA models in EMS are usually implemented using general-purpose programming languages (GPLs), such as FORTRAN, C/C++ or Java. Thus, the exact workings of the models might be disclosed to modelers trying to understand such models. Reasons for the use of GPL can be seen in the specificity of existing DSLs for cellular automata modeling and the monolithic character of modeling tools that provide DSLs. Tools for CA modeling that are based on extensions of GPL (libraries) do not address the issue of transparency, since important parts of the model specifications are specified by means of GPL. In contrast, DSLs for CA modeling that are independent of GPLs, might provide transparent abstract modeling concepts on the one hand, but on the other hand, these DSLs provide limited degrees of freedom, so that common environmental processes cannot be specified or their specification is rather complicated and, again, not transparent. It is a goal of this thesis to identify common characteristics of CA as used in EMS that can be formalized by an DSL that particularly takes aspects of pragmatics and transparency into account and thus contribute to a clarification of respective concepts (see Chapter 4).

EMS in general and CA-based modeling herein in particular requires in great parts the inclusion of data with a geospatial reference (geodata) in modeling and simulation. The integration of geodata and simulation is a current issue in the field of Geographic Information Systems (GIS) and typically discussed at the basis of the technology on which simulation software is based (e.g. programming languages, component technologies). It appears that with object-oriented metamodeling important standardization efforts in the field of GIS widely apply the same formal framework for describing data at an abstract level as do typical MDE-technologies for the description of modeling languages. However, a discussion of the feasibility of metamodel-based integration of GIS and simulation technology is not available. A goal of this thesis is the discussion of the integration of geodata and simulation modeling at the base of object-oriented language metamodels and the illustration of technical feasibility by means of a prototypical implementation of a respective DSL as a realization of main aspects of the above mentioned envisaged contributions and its application within an exemplary simulation study.

1.2 Research Goals and Methods

As mentioned above, this thesis is basically concerned with two fields of investigation. First, the feasibility of the application of MDE to EMS with particular emphasis on pragmatic aspects and language metamodeling. Second, the identification of a class of CA with particular consideration of those aspects that are related to EMS. Further, the topic of integrating geodata processing with simulation at the base of object-oriented language metamodels, which is necessary when applying corresponding MDE-technologies to EMS is a aspect under consideration.

At the most general level, the investigations within this thesis follow a common pattern. First, basic theoretical and technical aspects are considered at the base of a review of literature and tools. Second, for each field of investigation a concrete approach is formulated as the basis for a prototypical implementation, the application of which serves as a proof-of-concept. Thus, at the base of review of basic aspects of system-theoretic M&S and MDE, the Language-centered Approach (LCA) is formulated as a concrete application of ideas and technologies of MDE to EMS. Further, from a additional review of CA, the Environmental Cellular Automata Language (ECAL) is conceptualized and implemented within the framework of LCA. ECAL has been used to re-implement some models, that exemplify typical properties of CA as used in EMS. Moreover, ECAL has been used, in a comprehensive case study in the field of land-use change modeling in which a well-known approach (SLEUTH) has been re-implemented and modified, such that it adapts to data availability in the study region of Greater Tirana. The metamodel-based integration of DSLs and GIS is conceptualized at the base of review and has been implemented within the framework of LCA and is a constituting element of ECAL. The case study to which ECAL has been applied necessarily requires the integration of simulation and GIS.

The notion of "pragmatics" of computer languages is comprehensive in that it involves an integrated consideration of technical, epistemic and cognitive aspects. This thesis particularly focuses on aspects of pragmatics that are not considered to a adequate degree in related studies, that mainly focus on technical functional and non-functional properties of modeling and simulation software. For this, building upon corresponding well-accepted considerations of M&S, aspects derived from "model-based science" and "model-based reasoning" are particularly considered and set into the context of the particularities of MDE with language metamodeling and digital simulation. For this, relevant aspects of the notion of model-based science and model-based reasoning are characterized from literature review and the relation to well-accepted aspects of M&S and MDE is discussed.

1.3 Contributions

The first contribution of this work is the definition and evaluation of the Language-centered Approach (LCA) to EMS that is an the adaption of the model-driven engineering with metamodels to EMS with particular focus on the support for transparency and reusability of models. The first key aspect is the identification of requirements in the context of EMS and their relation to properties of MDE. The second key aspect is the identification of technologies and their underlying concepts that enable the application of MDE from a technical perspective. In particular, the integration of Geographic Information Systems with dynamic modeling and simulation which is based on object-oriented metamodels is necessary and the second contribution of this work. The third contribution is the formal

definition of a class of cellular automata models (environmental cellular automata (ECA) models) by means of a DSL (ECAL), where ECAL specifications describe specific types of processes whose formalization is usually realized with generic-purpose programming languages.

1.4 Outline

This text is structured as follows. Chapter 2 encompasses the review of basic system-theoretic, mathematical and methodological aspects of modeling and simulation in software engineering and EMS. This is followed by a characterization of basic aspects of computer languages and Model-driven Engineering in Chapter 3. A conclusion provides the identification basic pragmatic aspects in particular against the background of MDE and EMS presented in Chapters 2 and 3.

Chapter 4 presents Cellular Automata as typically used in EMS and aspects of Cellular Automata in general that relate to the former. In Chapter 5 the adaption of the model-driven approach to the context of EMS is presented by the definition of the Language-centered Approach (LCA) to tool support including the metamodel-based incorporation of geodata and geodata processing. Chapter 6 presents the Environmental Cellular Automata Language (ECAL) and its applications. The discussion and conclusion of this thesis follow in Chapter 7.

2 Environmental Modeling and Simulation (EMS)

Environmental modeling and simulation (EMS) refers to the application of the method of modeling and simulation (M&S) in environmental science and management. Although much of basic technology for M&S has been developed in the context of systems engineering, its perceived generality suggests application to other domains such as EMS. In general, M&S is characterized by the use of abstract mathematical models and computer simulation[1] for the purpose of knowledge discovery and knowledge application within the framework of systems theory. This chapter characterizes the fundamental terms and features of M&S, the specific characteristics of EMS and how these relate to environmental studies, in particular with respect to the role of the use of M&S technology and modeling languages herein. It is argued how tools for M&S can be understood as means to integrate epistemological-cognitive and technological aspects against the background of *model-based reasoning*. For this, this chapter proceeds as follows. After the characterization of environmental science and management (Chapter 2.1) the basic concepts for the conceptualization of systems in general and environmental systems in particular by means of system-theoretic dynamical systems is presented (Chapter 2.2). Chapter 2.3 presents common classes of models whereas the subsequent Chapter 2.4 describes their use against the epistemological-cognitive background of M&S as an experimental method and the role of modeling tools herein. Chapter 2.5 provides basic conclusions.

2.1 Environmental Science and Environmental Management

Environmental science is concerned with issues related to the environment. In a broad sense, *environment* is defined as the physical, non-living and living surrounding of organisms (Boersema, 2009). Environmental science is particularly concerned with the relation of the environment with humans and animals, coexisting in space and time, based on an interdisciplinary integrative view on physical, chemical, biological, social, political, economic and technological characteristics and processes (White et al., 1984; Vries, 2009). The major motivation for environmental science is, besides curiosity, the provision of knowledge in order to be able to deal reasonably with limited natural resources and influence the environment in a positive way (White et al., 1984). Thus, environmental science tries to enable informed environmental management and policy by providing scientific knowledge (Argent, 2005). Typically, this is related to issues of efficient and sustainable use of limited natural resources (e.g. food production), environmental services and protection (e.g. industry and pollution), disaster and risk management (e.g. natural hazards). Scientific considerations in environmental science are typically based on an evaluation of causalities within the environment and the consideration of actual and possible

[1] In this thesis the term *computer simulation* refers to computer simulation by means of digital computers as opposed to computer simulation by means of analog computers.

influence of human action. This involves the identification of causalities (e.g. natural laws), the assessment of the effect of human activities in the past (e.g. soil degradation through agricultural practices) and the evaluation of possible futures (e.g. with different policies). *Environmental management* is concerned with the influence of human action on possible futures for the purpose of being able to identify optimal measures. The concrete issues under consideration in environmental studies vary from the identification of specific relatively isolated single interrelations (e.g. influence of a specific farming technique to soil degradation) to relatively complex aggregate issues, such as the influence of a certain policy (e.g. Kyoto-Protocol) to public wealth. Management processes are characterized by the need to establish information flow and at best consensus among a number of stakeholders (e.g. scientists, politicians, affected people, Aumann (2011)). The credibility of information provided by simulation models, thus the degree to which encoded information is used effectively in management, largely depends on an agreement on the characteristics of simulation experiments (Aumann, 2011). Agreement typically results from a collective learning process, where experiments and simulation models are varied and discussed amongst stakeholders and aligned with personal mental models and goals (Aumann, 2011).

Environmental science and management encompasses a variety of disciplines, methods and goals, however there are three distinguishing characteristics that are commonly attributed to modeling and simulation in environmental science and management, that have major influence on the design of respective tools (see Chapter 2.2.4 for detail):

- a relatively high degree of uncertainty,
- a relatively high degree of complexity and
- an explicit geo-spatial reference.

The method of modeling and simulation (M&S) of dynamical systems is fundamental for both, knowledge discovery in environmental science and knowledge application in environmental management, where environmental models, represented by digital simulation models, are central artifacts (Beven, 2002; Jakeman et al., 2008; Brugnach et al., 2008; Argent, 2005) . Besides methodological aspects (see Chapter 2.4), it is an characterizing feature of M&S that its realization requires the implementation of computational simulation models that mimic the studied processes in order to facilitate experimentation (in contrast to typical laboratory experiments). The implementation of simulation models is technically such demanding that it is a well described scientific issue on its own (see Zeigler et al. (2000) for an overview). However, it is widely recognized that the implementation of "good" modeling processes requires more than mastering basic technical issues. The implementation of "good" modeling processes and adequate tool support is an important issue in environmental science and management, since to be able reach the goal of informed environmental management it is particularly important "that our development practices build social and scientific credibility" (McIntosh et al., 2008). In this context, the transparency of models has early been identified as a a major flaw of scientific M&S, particularly against the background of complexity and uncertainty: " ... probably the most important attribute any model should have is transparency. It should readily understandable to any potential user with a reasonable investment of effort" (Lee, 1975).

Although the importance of transparency is widely recognized, there is no common concrete understanding of it and there are few attempts to describe and relate both, the practical-technical issues and the cognitive-epistemological aspects of tool design for

EMS. In an attempt to fill this gap, the following chapters describe the background of M&S and EMS in order to clarify characteristics of "good" modeling processes and the role of transparency and M&S technology herein, against the background of relatively well-established mathematical and technical aspects.

2.2 System-theoretic Foundation of M&S in Engineering and EMS

General Systems Theory (GST) provides the basic conceptual framework for M&S as considered in this thesis (Chapter 2.2.1), since it is considered as the fundamental conceptual framework for M&S in general, and systems engineering with associated modeling languages in particular. *Dynamical Systems* provide a mathematical incarnation of fundamental aspects of GST, whereas Object-orientation takes a software-engineering perspective. Object-orientation has profound influence on the design of existing tools and is a fundamental aspect of MDE. Chapter 2.2.1 gives an overview of the basic concepts of GST followed by a characterization of most basic properties of Dynamical Systems (Chapter 2.2.2) and Object-orientation (Chapter 2.2.3). Chapter 2.2.4 specifies characterizing features of EMS that have influence on the design of modeling tools for EMS.

2.2.1 General Systems Theory

GST, as first presented as such in Bertalanffy (1950), is an approach to scientific knowledge discovery which is - in contrast to "mechanistic view" - based on the assumption that it is generally not possible to explain all phenomena by means of basic physical natural laws. GST rather assumes that explanations may be based on the characterization of *systems*, which are entities that structure the perception of real world phenomena and that are meant to follow system-specific principles. It is a distinguishing feature of GST that systems and their principles may be defined at any level of abstraction, such that systems and respective principles might be generalized and applied to a number of real world phenomena. By this general systems and respective general principles (e.g. exponential growth), comparable to basic laws in physics, might be identified in scientific disciplines other than physics.

In the context of system-theoretic M&S, *system*, *model* and *observation* constitute basic concepts that are defined and related within empirical studies. Usually, such studies follow a general procedural template: in the first step of a study the problem to be solved or the hypothesis to be tested is stated and objectives of the study are derived from that. Based on these objectives, a clearly defined part of reality is identified for examination and designated as a *system*[2]. The part of reality that does not belong to the system, but may have relevant influence on it, is declared as the *system environment*, separated from the system by the *system boundary*. A basic distinction is made between *open systems*, that interact with the system environment and *closed systems* that have no relevant interaction with the environment.

Bossel (2004) identifies three basic defining properties of systems:

- *Goal-directedness*: A system provides specific functions in order to accomplish a specific goal.

[2]In literature the term *real system* is used in order to distinguish *system* in the sense of "part of reality" from *system* in the sense of "system of equations".

- *Structure*: A system is composed of system elements according to a system-specific structure that enables the provision of the system's function.
- *Integrity*: The removal of system elements impedes the provision of the defining functions of a system.

Figure 2.1 presents the basic system-theoretic concepts and illustrates that a system is both, a conceptual framework and a source of observations through which reality is perceived. The perceived structure and observations of a system are attributed to reality, thus regarded to be "true" features of it (Zeigler, 1984). Observations are the base for the conceptualization of systems and corresponding specification of models. Models are a fundamental tool for the documentation of a system's perceived structure and the refinement of perceived structures beyond the directly observable, since models allow interaction and modification (see below), which systems often do not sufficiently.

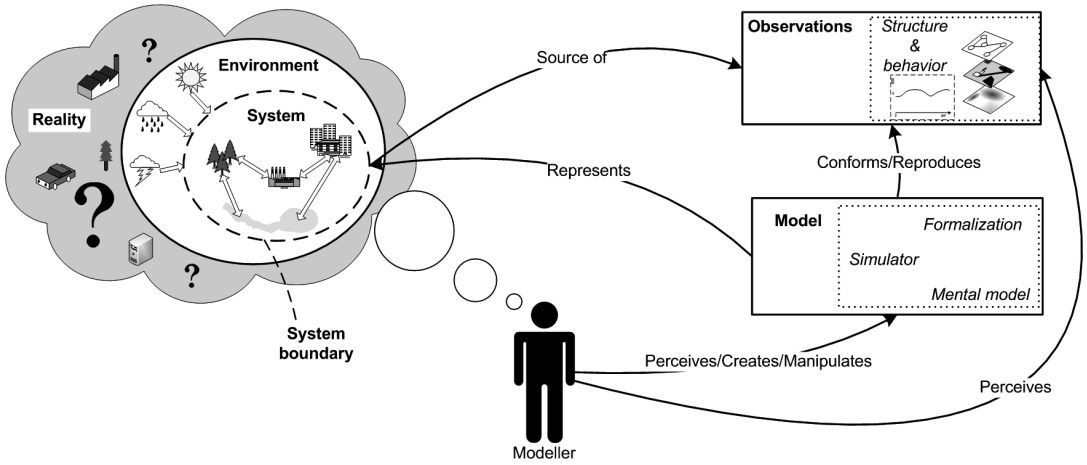

Figure 2.1: System, model and observation in the context of system-theoretic M&S.

According to GST the conceptualization of systems adheres to a specific forestructure: based on the perception that "reality [...] appears as a tremendous hierarchical order of organized entities, leading, in a superposition of many levels, from physical and chemical to biological and sociological systems", the systems approach follows the idea that "the system under study is decomposed into parts and relations of organizations [systems]" (Bertalanffy, 1950). Thus, a system is conceptualized as a complex of interacting *system elements*, where system elements themselves might be systems at the next lower level of abstraction - further referred to as sub-systems. However, at the lowest level of containment hierarchy a (sub-)system must be composed of basic system elements that are not sub-systems. Besides intuitive conceptualization, the identification of sub-systems allows for separate investigation of system elements. The behavior of the system follows from the behavior and interaction of contained system elements (Bertalanffy, 1950; Dilworth, 2009).

An inherent property of the notion of "system" is that it embodies properties of reality that are thought of being time-invariant and properties that vary with time. A characterization of a system at a particular time is usually referred to as the *state of the system*, which we can, at least partly, observe. In a given time span, variations of a system's state are thought of being constrained or caused by the system's time-invariant properties and

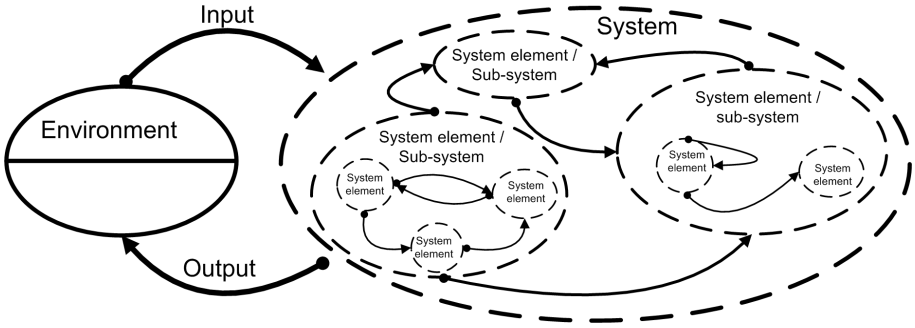

Figure 2.2: Hierarchical structure of systems.

the influence of the system environment[3]. However, only the characteristics of the system under consideration, e.g. the internal generation of behavior or the response to input, not the internal characteristics of the environment, are thought of creating the relevant behavior. Thus, the system - environment interaction must not contain feedback loops. In contrast, feedback loops between system elements are often perceived as a major source for a system's dynamic behavior (see Figure 2.2).

Experimentation, thus the controlled variation of a system's environment and observation of the respective systems' state, is a common approach to derive properties of systems (see more detail in Chapter 2.4). The limited accessibility of systems and their environments with respect to observation and manipulation often requires the use of *models* instead of real systems for the realization of experiments. In literature, term "model" is used in many different meanings, thus it needs clarification. A model might be generally defined as an "abstraction of a system intended to replicate some properties of that system (Overstreet and Nance, 1985)". The properties to replicate depend on the purpose of the study of which the experiment is part of. The respective abstractions are generally directed towards fulfilling the objectives of a study, thus the corresponding models are generally specific objectives of a study. Depending on the context, in literature the term "model" usually refers to one or more interrelated model representations of one of the following types of model representation:

- *Model specification*: a human readable, often formal mathematical, representation of a model based on a textual or graphical notation.
- *Simulator*: a mechanical representation of a model used for generating the dynamical behavior of a model. Generally there is a distinction between simulators that directly embody relevant properties of a system (e.g. scaled physical model) and digital computers as simulators that mimick a system in terms of computations (see Chapter 2.4)[4].
- *Mental model*: representation of a model as it appears in the mind of modelers[5]

[3]The state of the system is typically composed of those properties that are essential for the system's behavior in contrast to those that are a result of these essential properties. Against the background of uncertainty, the knowledge about status of properties might not be clear, such that this distinction is not to be made a-priory in general, but follows from investigation.

[4]This thesis is solely concerned with *digital systems analysis* where the simulator is realized on the base of a digital computer. In contrast to simulators that are scaled physical models, simulators based on computers pose specific requirements on model representation and interaction (see below).

[5]In literature the term "mental model" may refer to the actual assumed format information is stored in

often also referred to as *conceptual model*[6] in systems analysis.

Through their representations models are relatively easily accessible for creation, modification and observation. During the course of a study, usually several interrelated representations of a model exist in parallel (see section 2.4.1 for detail). Against this background, the term model is defined as follows in this thesis:

> *A model is a abstract simplified conceptualization of a system for the purpose of derivation of statements about the system. At any time of its existence, a model is represented by at least one concrete representation of the type "model specification", "simulator" or "mental model".*

The structure of a model might follow the perceived structure of a system with hierarchically composed models corresponding to hierarchically composed systems. Thus, depending on usage context, a model representation may be perceived as a representation of a system model, or a sub-system model or a model of an system element. The specific possibilities of representation of hierarchical models (e.g. feedback) indeed depends on the specific form of representation (e.g. the modeling paradigm, see Chapter 2.3.2).

A major goal of system theoretic modeling is the identification of structures, principles and laws that apply to a range of systems which are possibly attributed to different domains or scientific disciplines (Bertalanffy, 1950). Thus, system theoretic modeling not only aims at finding system-specific structures, but also at their generalization (e.g. by means of general laws), that sensibly apply to several systems or objectives. Against this background, the term "hierarchy" refers to two types of hierarchies in the context of GST: first, the compositional hierarchical ordering of systems and, second, the hierarchical ordering of concepts that denotes a classification of systems with respect to perceived structural and behavioral characteristics. Both, the identification and representation of composition and generalization hierarchies is essential part of system-theoretic studies. From the variety of possibilities, this thesis is concerned with a specific type of mathematical representations that are most relevant for M&S (see next Chapters).

The environment of the highest-level system is typically, amongst others, conceptualized in terms of the experimental setup, that at least refers to the system's inputs and outputs of the highest-level model (see Chapter 2.4.1 for more details). Classification hierarchies are elaborated further in Chapters 2.2.3 and 2.3.

Specific kinds of conceptualization of systems and experiments are presented in the following chapters.

2.2.2 Mathematical Dynamical Systems

The application of General System Theory in the sciences is closely related to the theory of mathematical *Dynamical Systems*. Dynamical Systems are way of mathematically modeling the evolution of processes with time[7]. The basis of a Dynamical System is a *phase space* (or state space) that represents possible states of the Dynamical System, *time* and

the brain. This thesis follows the notion of a "mental model" which refers to a relatively higher level of abstraction, e.g. referring to an imagined mechanism.

[6] In literature, the term 'conceptual model' often refers to "the model as it exists in the mind of the modeler"(Schruben and Yücesan, 1993) or more restrictively as "a series of mathematical and logical relationships concerning the components and the structure of the system" (Banks, 2000).

[7] In M&S literature, term "system" is used in different meanings. In mathematics, "system" often refers to a mathematical system as a set of assumptions and laws (Chorafas, 1965). In the context dynamical

an *evolution law* that is, a rule that allows for determination of the state at time t from the knowledge of the states at all previous times (Boccara, 2004). There are numerous ways to specify Dynamical Systems, some of which are characterized in the following Chapters after some more general characteristics of Dynamical Systems are sketched.

Figure 2.3 illustrates some fundamental aspects of Dynamical Systems. Given an input for a time span under consideration, an open system is perceived to produce a corresponding output. Observed or assumed variant characteristics of systems are represented by means of a set of variables, where each variable represents a specific time-variant property of the modeled system (e.g. a basic system element) . Corresponding time-invariant properties may be represented by means of parameters and invariant relations.

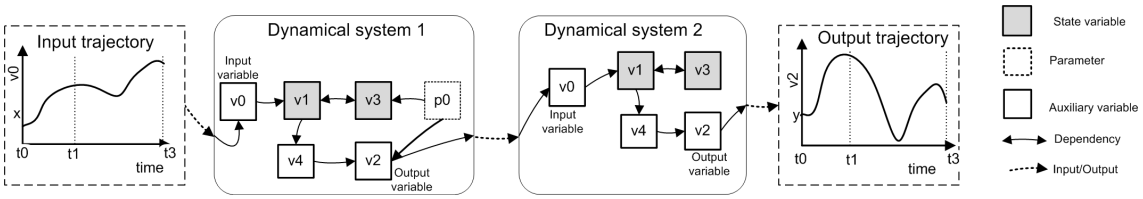

Figure 2.3: Illustration of the structure of dynamical systems.

The total set of variables that describe a Dynamical System is referred to as the set of *descriptive variables*. For each Dynamical System a subset of descriptive variables is identified as the set of *state variables*, that uniquely determine the current and future value of all descriptive variables, where the value of state variables cannot be derived from the value of other variables. However, state variables (i.e. *v1*, *v3*) may be interdependent. The set of state variables may not be unique in that it might be possible to identify different sets of state variables from a set of descriptive variables. In practice, the set of variables is usually not minimal since it typically contains dependent auxiliary variables (e.g. random seeds, variables for intermediate values) that are determined externally or immediately determined by the state variables (Zeigler, 1984).

In the context of practical system-theoretic application of Dynamical Systems in M&S with a compositional structure of open systems, the notion of input and output of a system is typically stated explicitly: Variables that represent the influence of the system's environment are designated as *input variables* and variables that represent the influence of the system on the environment are designated as *output variables*. The observed evolution of the values of variables of a Dynamical System with time in a given time span is referred to as the *trajectory*, where the evolution of state (*state trajectory*[8]) may be explicitly distinguished from the evolution of input (*input trajectory*) and output *output trajectory* (e.g. Zeigler et al. (2000)).

At a general level, a system-theoretical Dynamical System may be represented as a structure S:

$$S = \langle time, X, Q, Y, \delta, \lambda \rangle$$

, where:

systems theory the term "system" often refers to an ensemble of nonlinear equations (Boccara, 2004). In the terminology of this thesis, the term "system" in "Dynamical System" refers to a model (specification) of a (real) system.

[8]The state trajectory is often referred to as the "orbit" of a system.

- *time* is the time base $\langle T, < \rangle$, where T is the set of possible points in time and $<$ is an ordering relation. Typically, $t_0 \in T$ is given as the minimal time.
- X is the set of possible inputs.
- Q is the set of possible states. Typically, $q_0 \in Q$ is given as the state at time t_0 (initial state).
- Y is the set of possible outputs.
- $\delta : Q \times T \times X \to Q$ is the state transition function,
- $\lambda : Q \times T \times X \to Y$ (or $\lambda : Q \times T \to Y$) is the output function.

X, Q and Y may be thought of as being defined by vectors (input vector, state vector, output vector resp.), where a vector component may be thought of referring to a variable[9]. Each vector component is associated with a set of possible values with the total set of possible input, state and output values being the respective crossproduct[10]. In this text, vectors with specific values are referred to as x or $\underline{x}$ ($x \in X$), q or $\underline{q}$ ($q \in Q$) and y or $\underline{y}$ ($y \in Y$) and values of components x_i ($x_i \in X_i$), q_j ($q_j \in Q_j$) and y_k ($y_k \in Y_k$), where i,j and k refer to the ith, jth, kth component, respectively. Alternatively, the value of variables may be referred to by means of their names, which then is clear from context.

An input trajectory provides the value of input (x) for every point in time of interest $t_n \in T$, the output trajectory provides the corresponding output (y), the state trajectory the value of q, respectively. Each trajectory can be divided into contiguous segments, where each segment is associated with a value for x, y and q and two times ($\langle t_n, t_{n+1} \rangle$) that denote the beginning (t_n) and end (t_{n+1}) of a segment. Segments are contiguous if the time of the time of the beginning of a segment equals the time of the end of the former segment. Trajectories are perceived as the concatenation of contiguous segments starting at time t_0.

Different types of segments are associated with different classes of Dynamical Systems and corresponding forms of specification, in particular transition functions (see Chapter 2.3.1). A common characteristic of the specification of Dynamical Systems is that the change of state (δ) is basically specified in terms of increase or decrease of values of state variables within segments, such that states can be calculated for all times that determine contiguous segments $t_n \in T$, thus the trajectory, given an initial configuration (q_0) and, if needed, inputs. Generally, increase (decrease) is either given in terms of rates of change (e.g. $dq/dt = f(q(t), x(t), t)$) for continuous Dynamical Systems or discrete state changes (e.g. $q(t_{n+1}) = f(q(t_n), x(t_{n+1}), t_{n+1})$). Historically, the definition of Dynamical Systems and characterization of interesting properties is closely related to the use of differential equations and difference equations for their description.

According to the compositional structure of systems, Dynamical systems might be specified as a coupled network of Dynamical Systems, where each component provides a partial specification of the composed system (N) that are related by means of input-output couplings. A coupled Dynamical System may take the general form (see Zeigler et al. (2000)):

[9] Formally, $V^{in} = (v_1^{in}, v_2^{in}, ..., v_n^{in})$, $V^{state} = (v_1^{state}, v_2^{state}, ..., v_m^{state})$, $V^{out} = (v_1^{out}, v_2^{out}, ..., v_l^{out})$, with V denoting a set of variable names (v) and n, m and l the respective number of variables.

[10] Formally, if sets of possible values are denoted by $X_1, X_2, ..., X_n$ for input variables, $Q_1, Q_2, ..., Q_m$ for state variables and $Y_1, Y_2, ..., Y_l$ for output variables, the resulting sets of possible values are $X = X_1 \times X_2 \times ... \times X_n$ for input, $Q = Q_1 \times Q_2 \times ... \times Q_m$ for state and $Y = Y_1 \times Y_2 \times ... \times Y_l$ for output.

$$N = \langle time, X_N, Y_N, D, \{M_d\}, \{I_d | d \in D \cup \{N\}\}, \{Z_d | d \in \cup\{N\}\}\rangle \quad (2.1)$$

where D is a set of *component references*, $\{M_d\}$ the set of components[11], each contributing a partial specification of the state and the state transition function of N. I_d and Z_d specify input-output mappings of components (see Zeigler et al. (2000) for elaboration)[12]. Indeed there are restrictions on possible couplings (e.g. possible ranges of values and the time flow mechanisms must match, see Zeigler et al. (2000)) and there is a variety of ways to define components and couplings (e.g. typed ports) each with specific constraints (see Chapters 2.3, 3.3, 3.4). Zeigler et al. (2000) provides a discussion of respective formalisms and constraints.

Besides mathematical rigour, Zeigler et al. (2000) and Klir (2003) argue that a fundamental aspect of system-theoretic Dynamical Systems is the degree to which relevant knowledge about a system is encoded in its specification. Such encoding of knowledge ranges from mere description of inputs and outputs at the system boundary (resp. X, and outputs Y, without Q, δ and λ) without causalities to the specification of a Dynamical Systems as "generative mechanisms", where the internal causal structure is specified by means of Q, δ and λ or even as a system of coupled Dynamical Systems that "generate" observed behavior, instead of merely replicating observations. The specification of causal relationships allows the specification of initial value problems, where the invariant properties of a Dynamical System, the first state (initial state, q_{t_0}) and inputs (x) are given and trajectories are derived at the base of δ and λ[13]. This approach is a basic ingredient of the experimental approach for the identification of causal structures of systems (see Chapter 2.4).

The identification of causal structures and their specification by means of models (Dynamical Systems) is the outcome of studies as a process of knowledge discovery and at best involves generalization and the explicit mathematical specification of generalizations. There are numerous ways of specifying Dynamical Systems and according initial value problems (see Chapter 2.3.2). Dynamical Systems in this text refer to Dynamical Systems that are specified by means of generative mechanisms (including Q, δ and λ), typically as initial value problems.

A fundamental aspect of the investigation of Dynamical Systems in general and the application of Cellular Automata in particular is the consideration of complexity that is particularly related to the notions of non-linearity and chaos, which pose particular methodological challenges. The theory of Dynamical Systems is traditionally concerned with the long-term behavior of Dynamical Systems as specified by means of differential equations and difference equations (see Chapter 2.3). Some basic types of Dynamical Systems are commonly distinguished at the base of their long term behavior, thus the state(s) reached as time goes to infinity (limit sets). Equilibria (e.g. stationary points $\delta\prime = dx/dt = 0$, fixed points $\delta(x^*) = x^*$) and periodic points ($\delta^\tau(x) = x$, $\tau \in \mathbb{I}^+$) and their characteristics with respect to stability are of particular interest (see Boccara (2004)). Analysis allows the identification of attractors or repellers, that is fixed sets of equilibrium states to which a system evolves from neighboring states (attractors, asymptotically stable)

[11] Components are specified as Dynamical Systems $M_d = \langle time, X_d, Y_d, Q_d, \delta_d, \lambda_d\rangle$

[12] I_d is the set of components that influence component d and Z_d is a mapping of respective outputs and inputs in an adequate form (e.g. variable-to-variable mapping).

[13] Please note that "application of δ" may refer to actual iterative application, but this also encompasses analytical or numerical derivation of properties (see Chapter 2.3.2).

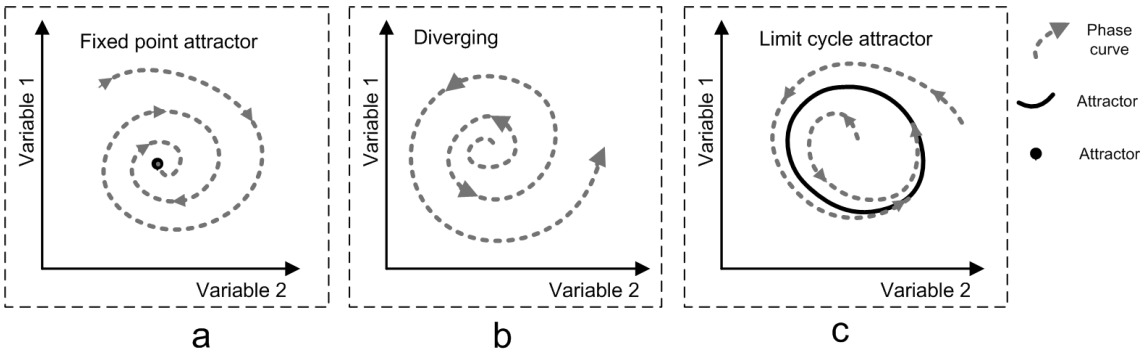

Figure 2.4: Behavioral classes of dynamical systems and their attractors.

or from which a system moves away (repeller, unstable). Figure 2.4 illustrates three basic prototypical behaviors at the base of exemplary phase portraits. In the most simple prototypical cases, the system either grows infinitely (b) or it evolves towards a fixed point attractor (a) or limit cycle attractor (c) that consists of periodic points[14]. The presented types of long term behavior have in common a certain degree of predictability, in the sense that given a mathematical description, it is possible to derive interesting properties (e.g. behavioral properties) with relative ease (see Chapter 2.3.2).

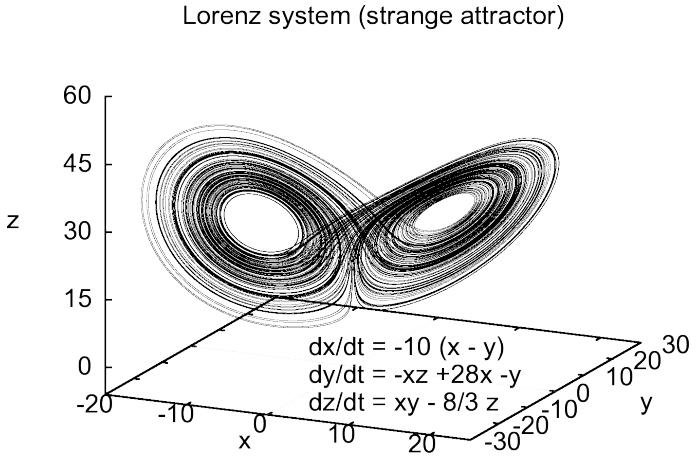

Figure 2.5: Strange attractors of Lorenz system.

In contrast, *chaotic systems* appear to be relatively intricate, but they are particularly relevant with respect to complex systems and the application of Cellular Automata (see Chapter 4). Chaotic systems are characterized by an apparently unpredictable and seemingly random behavior, while being fully deterministic. Figure 2.5 illustrates chaotic behavior at the example of the *strange attractor* of the Lorenz system, which evolves infinitely on the attractor, that only takes a limited portion of phase space, while never visiting the same state twice, thus never reaching a stable state or cyclic behavior. The motion of a chaotic system appears to be random in that it appears to be impossible to

[14]Please note that more behaviors are possible (e.g. quasiperiodic behavior, Lyapunov stability) and that limit set are commonly further distinguished (e.g. node and focus according to type of attraction and repulsion), which is not presented here (see Boccara (2004))

to predict which path the system takes in particular in the long run.

There are various competing mathematical definitions of the notion of "chaotic system" currently under discussion (see Blanchard (2009)), however the characterization presented in Boccara (2004) explains the basic notions with adequate level of sophistication for the purpose of this thesis. Boccara (2004) (with reference to Devaney (1989)) provides a rather intuitive notion of chaotic system is mainly characterized by three properties:

- unpredictability,
- indecomposability and
- an element of regularity.

"Unpredictability" is associated with the *sensitive dependence on initial conditions*, which refers to the fact that a arbitrarily small deviance of the initial state might lead to uncorrelated behavior with time[15]. Thus, prediction generally requires unlimited precision when encoding initial states and numerical treatment, which both is not available in practice, thus there is unpredictability. Figure 2.6 illustrates sensitivity at the example of two trajectories of the Lorenz system with a small deviation of the initial condition ($\Delta y_0 = 0.005$). Trajectories show similar evolution at small t (gray), before they eventually diverge and appear uncorrelated at $t = 15$ (black)[16].

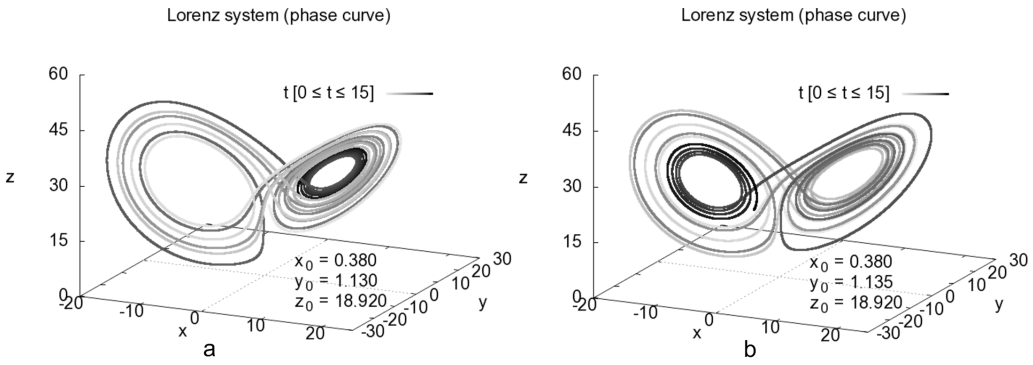

Figure 2.6: Two trajectories of Lorenz systems with slightly different initial states and diverging trajectories.

"Undecomposability" refers to the characteristic that a chaotic system might move from any neighborhood of a state to any other possible state, thus the system might not be decomposed into subsystems, in the sense that distinct subsets of the state space can be identified on which δ operates independently. Formally, this notion can be expressed by the characteristic of *topological transitivity*[17].

[15] Formally, sensitive dependence on initial conditions may be defined for closed system in that δ is such that there exists a sensitivity constant $\theta > 0$ such that for any $q_1 \in Q$, $\epsilon > 0$, there exists $q_2 \in Q$ with $d(q_1, q_2) < \epsilon$ and n such that $d(\delta^n(q_1), \delta^n(q_2)) \geq \theta$, d being a distance (see Blanchard (2009)).

[16] Please note that trajectories themselves, thus the deviation may also be an artifact of the precision of the numerical method chosen and precision of digital representation of numbers (i.e. Euler forward integration with $\Delta t = 0.0001$ and 64-bit floating point number representation with 15 significant digits in Figure 2.6).

[17] Formally, Q cannot be decomposed into different open sets that are independent under δ: for any pair of two open sets (U, V) in Q there exists $n \in \mathbb{N}^+$ such that $\delta^n(U) \cap V \neq \emptyset$ (see Blanchard (2009)).

The "element of regularity" is that periodic points are dense in Q, meaning that there is an infinite set of periodic points, where each state $q \in Q$ either is part of a periodic trajectory or it is arbitrarily close to a periodic point[18]. For chaotic systems these periodic trajectories are not stable, thus not attractive or there is zero probability to exactly match an attracting periodic trajectory (limit cycle) in practice (Townsend, 1992).

Another property that is associated with chaotic Dynamical Systems is that chaotic behavior depends on the value of parameters, thus the same structure encoded by the Dynamical System may exhibit different types of long term behavior depending on the value of parameters. This "road to chaos" is associated with a change of stability of equilibria of corresponding difference and differential equations as the control parameter is varied (see Boccara (2004))[19]. Parameter values and equilibria at which long term behavior changes qualitatively (e.g. leading to a different type of attractor) are referred to as *bifurcation points*. Bifurcation points at a critical transition value r_T are characterized by changes of the stability of solutions (e.g. in terms of eigenvalues of the Jacobian) such that type of limit behavior is different for $r < r_T$ and $r > r_T$ (see Boccara (2004)).

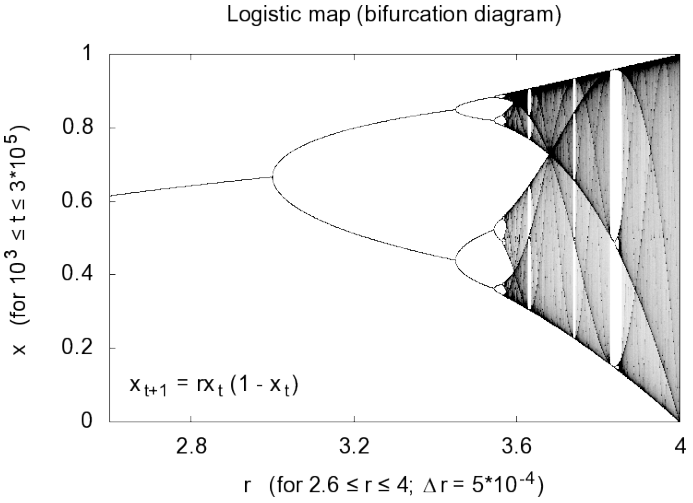

Figure 2.7: Bifurcation diagram of the logistic map.

Figure 2.7 illustrates bifurcation at the example of an bifurcation diagram for the *Logistic map* ($x_{t+1} = rx_t(1 - x_t)$). The bifurcation diagram shows the values of a state variable (here x) that are reached by a system in the long run for a given parameter value (here r). Shades of gray indicate the density of points, thus solid lines may refer to fixed points or limit points and white areas are not reached in the long run. For values of $r \leq 3.0$ the system evolves towards a fixed point, for values $3 \geq r \geq 1 + \sqrt{6}$ (≈ 3.45), the system oscillates between two values in the long run, for values $\approx 3.45 \geq r \geq 3.54$ the system oscillates between four values. With further increasing r more "period doublings" occur with decreasing distance along r until infinite period doubling and chaotic behavior at ≈ 3.599692. This is referred to as *period doubling route to chaos*.

Figure 2.8 presents trajectories of the logistic map for different values of r, with fixed

[18]Formally, let $P(\delta) = \{q \in Q | \exists n \in \mathbb{N} : F^n(q) = q\}$ be the set of periodic points of δ has dense periodic trajectories iff $P(\delta)$ is a dense subset of Q, i.e. for any $q \in Q$ and $\epsilon > 0$, there exists $q* \in P(\delta)$ such that $d(q, q*) < \epsilon$ (see Margara (1999)).

[19]Stability refers to the characteristic of solution, that a system remains in the vicinity of x^*in case of perturbations (stable) or diverges (unstable).

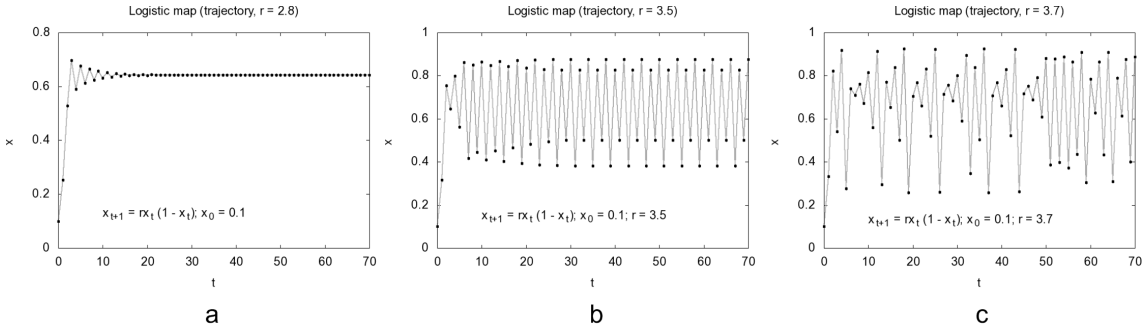

Figure 2.8: Trajectories of the logistic map for different values of r.

point long-term behavior for $r = 2.8$ (a), periodic behavior for $r = 3.5$ (b) and chaotic behavior for $r = 3.7$ (c). Other roads to chaos have been identified, e.g. the critical transition value $r_T = 1 + 2\sqrt{2} \approx 3.828427$ exemplifies the *intermittency route to chaos*, where periodic behavior for $r < t_T$ changes to seemingly periodic behavior that is interrupted by unpredictable bursts of chaotic behavior for $r > r_T$ (see Boccara (2004) for illustration). Depending on the stability characteristics of solutions around bifurcations points different types of bifurcations have been identified (e.g. saddle-node bifurcation, Hopf bifurcation etc.), which are typically characterized by stability properties of equilibria of respective difference equations or differential equations (see Boccara (2004)).

Chaotic Dynamical Systems pose specific challenges to M&S studies in that modeling and simulation is particularly intricate, however chaotic systems are particularly relevant for EMS. Against this context Cellular Automata have a history of application to chaotic systems and systems "at the edge of chaos", due to particular properties of CA, as opposed to other modeling paradigms (e.g. differential equations, see Chapter 2.3 and 4).

The following chapter presents object-orientation as a way to conceptualize and formalize models as Dynamical Systems with particular focus on digital computing. Object-orientation is of particular relevance since it bridges digital simulation with modeling with Dynamical systems and it is also a fundamental ingredient of the approach to define modeling languages that subject of this thesis. This is followed by a characterization of specifics of EMS and further elaboration of investigation of Dynamical Systems.

2.2.3 Object-oriented Systems

Object-orientation refers to an approach to software engineering that resembles basic system-theoretic considerations and which is highly influential to the design of programming languages and tools for M&S, thus object-orientation combines system-theoretic modeling with software development (see Chapter 3.3). As a widespread and general approach to describe compositionally and conceptually hierarchically structured Dynamical Systems, object-orientation provides general concepts that offer themselves for inclusion in DSLs. Further, object-orientation provides basic conceptual and technical background for the definition of metamodel-based computer languages which is a defining element of the approach that is evaluated in this thesis (see chapter 3.5). Basic characteristics of object-orientation are in the following presented as described in Booch (2004).

Like software engineering in general, object-orientation is concerned with issues arising from complex problem domains and implementation technologies in combination with

issues of project management (e.g. heterogeneous institutional settings, changing requirements, limited resources)[20]. Against this background, object-orientation is to be understood as a comprehensive development method that encompasses the identification of requirements (Object-oriented Analysis), the conceptualization (Object-oriented Design) and implementation (Object-oriented Programming) of software systems (Booch, 2004). This thesis however is mainly concerned with with the most basic conceptual and technical properties of object-orientation, not the associated system development method.

The basic concepts of object-orientation are prescribed in the *object model*. The object model states that the system under study is conceptualized as consisting of interacting objects, thus the real system is represented by a collection of objects and their relations.

> *An object is a tangible entity existing in space and time with which one can interact[21]. An object has a well defined state, behavior and an identity.*

The state of the system is composed of the state of the corresponding objects according to the structure that is formed by the relations between objects. Both, relations (structure) and the objects' state may change as a result of interaction.

Figure 2.9 illustrates the idea of nested objects having various possibilities of interaction.

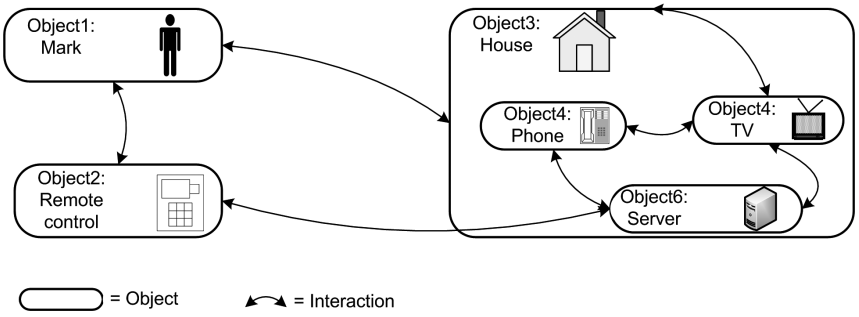

Figure 2.9: Object-oriented conceptualization of a system, where interacting objects are arranged in a containment hierarchy.

The state of an object is represented by means of typed attributes (e.g. numerical variables) the possible behavior of an object is modeled by means of methods. For modeling the object structure, Booch (2004) identifies two main types of relationships between objects: object links and object aggregation. Object aggregation refers to the feature that an object may contain other objects, so that it is possible to navigate from a container object to the objects contained in it, and vice versa. This allows determination of the state of an object by recursively composing the state of an object of the state of all contained objects. Booch (2004) refers to this containment hierarchy as *object hierarchy*. In contrast, a link denotes the interaction between objects or more specifically " [...] the specific association through which one object (the client) applies the services of another object (the supplier), or through which one object may navigate to another" (Booch, 2004). The state of the system - in the sense of state variables of a Dynamical System - is the

[20] Although object-orientation originates in the field of simulation (SIMULA), its major development has taken place in the field of software engineering, where different variants of object-orientation have been developed (Sebesta, 2004).

[21] In this context, objects exist in digital computers in some medium (disk, memory etc.).

composition of the state of all objects and the structure is represented by object links and aggregation. The dynamic behavior of the system is the result of interaction that occurs as a result of the execution of invocations operations along the flow of control that is passed across objects as methods are invoked.

A further defining characteristic of object-orientation is the explicit representation of a generalization hierarchy that defines object-oriented models. An object-oriented *class hierarchy* denotes an arrangement of abstractions that are used to model object hierarchies.

> *A class is an abstraction that can be used to instantiate objects. It represents common structural and behavioral characteristics of possible instantiations.*

A class hierarchy basically consists of a number related classes, where a class prescribes possible properties and relations of those objects that are "instances"[22] of the respective class: possible values of attributes, possible behaviors (operations) and relationships (links, aggregation).

Classes themselves are related by means of inheritance relationships, according to which a more abstract class represents properties that are common for all subclasses, that refine or extend properties of superclasses (see below). Whereas aggregation and links provide templates for the instantiation of the object-hierarchy, inheritance provides the means to express a generalization hierarchy of corresponding concepts.

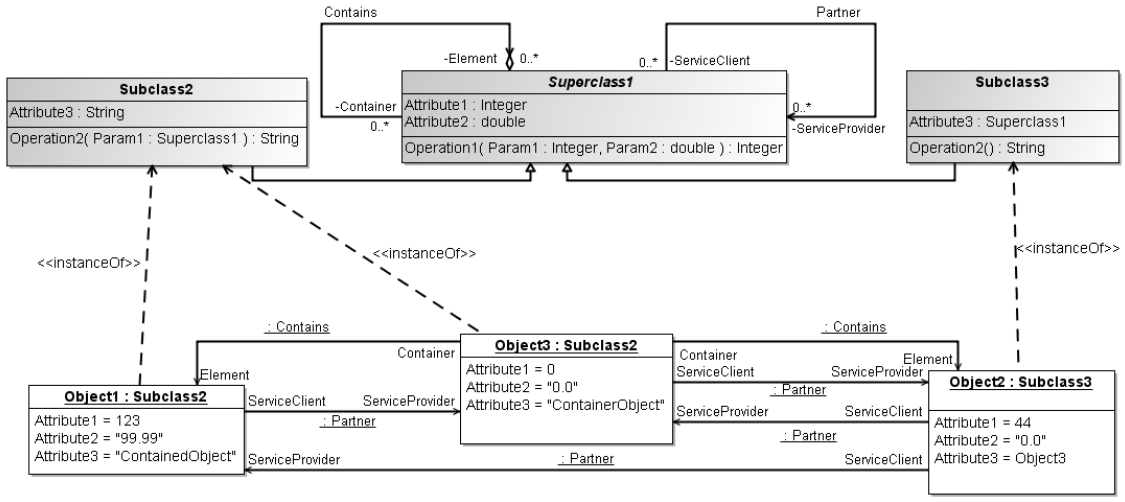

Figure 2.10: A simple object-oriented class and object-hierarchy (UML notation).

There are various ways to concretize and realize the notion of object-orientation. An exemplary and widely adopted approach is that of Unified Modeling Language (UML). Figure 2.10 presents basic features of object-oriented class and object hierarchies in terms of UML notation (see Booch (2004)) at the example of illustrative - though with regards to content meaningless - class and object hierarchies. The classes (gray boxes, i.e. *Superclass1*) define the possible state of objects by means of typed attributes (i.e. *Attribute1* ...), which are variables, if dynamic, and constants, if static. Objects (white boxes, i.e.

[22] Each object hierarchy is a materialization of a class hierarchy that is commonly associated with the process of *instantiation* (e.g. allocation and population of computer memory etc.) in the context of object-oriented programming with object-oriented programming languages.

Object1 ...) are instantiations classes (denoted by *«instanceOf»*) with concrete instances of states (i.e. *Attribute1 = 123* in *Object1*). A subclass inherits characteristics (variables, constants, relationships and operations) from its superclass (i.e. subclasses *Subclass2* and *Subclass3* inherits *Attribute1*, *Attribute2* and *Operation1* from *Superclass1*). Moreover, a subclass might augment or restrict the superclass by addition or by redefinition of characteristics (i.e. *Attribute3* and *Operation2* in *Subclass2* and *Subclass3*). A concrete class with all properties specified, can be used as a template to instantiate a number of structurally and behaviorally similar, but not identical, objects (i.e. *Subclass2* is used to instantiate *Object1* and *Object3*).

Please note that object-orientation also incorporates the notion of *active classes*, which are classes with that may exhibit behavior irrespective of being invoked by a method. Thus, there might be a number of corresponding interacting active objects that evolve in parallel. Further, through instantiation, the number of objects might vary according to the dynamics, which implies that, when perceived as a Dynamical System, object-orientation allows the number of state variables and the structure of the system to change.

The aggregation *Contains* denotes that, as a *Container*, each instance of *Superclass1*[23] may contain other instances of the same type as *elements* (and that each instance might be contained in several other instances). The link[24] *Partner* prescribes that any instance of *Superclass1* may link to any other instance either as a provider of services (*serviceProvider*) or as a client (*serviceClient*). Provided services are defined by means of operations of a class (i.e. *Operation1* in *Superclass1*). Such operation is thought to be invoked by a client that sends a message to the provider of the operation. The invocation of an object's operation may change the provider object's state and may return a message with some content to the client, that may be used to change the client's state. The state change and return value of an operation depend on the state of the provider object and the content of the message sent by the client object.

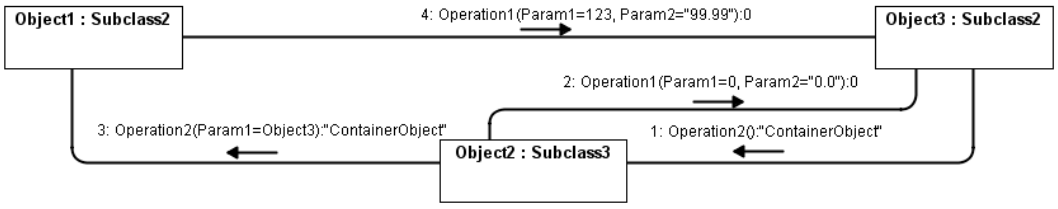

Figure 2.11: Interaction between objects (UML-notation)

Figure 2.10 denotes a possible instantiation of containment and interaction through connections between the objects (i.e. *Object3* contains *Object1* and is a *Partner* of *Object1* as a *serviceProvider*.) Figure 2.11 denotes a possible sequence of messages that invoke respective operations: First, *Object3* invokes *Operation2* at *Object2* then *Object2* invokes

[23] Indeed the abstract class *Superclass1* cannot be instantiated directly, but via its concrete subclasses *Subclass2* and *Subclass3*.

[24] Links are denoted as associations represented as connecting lines.

Operation1 at *Object3* etc.

Typically, related classes may be grouped into modules (e.g. packages in Java/UML). Modularization supports the reuse of conceptualizations through the relation of packages as illustrated in Figure 2.12. Given that classes in Figure 2.10 reside in package *Package1*, the definition of another module (i.e. *Package2*) directly uses classes from *Package1*. I.e. a new class could be defined as a subclass of *Superclass1* from *Package1*, with modifications (i.e. *Attribue3*), such that *Object4* can be instantiated from it[25].

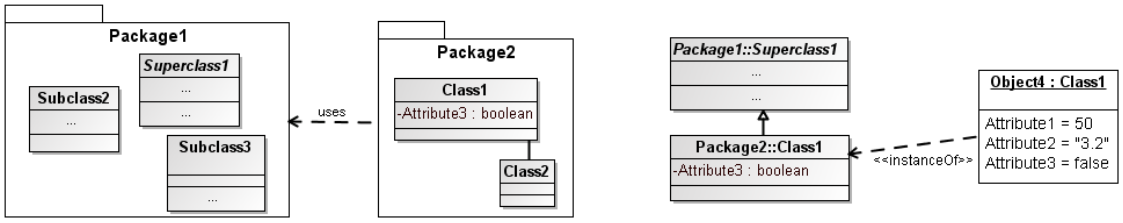

Figure 2.12: Modularization of object-oriented conceptualizations at the example of UML packages.

In practice, both, the object hierarchy and the class hierarchy need to be formally defined at some point of the development process to be of use as a base for software system implementation and realization of supporting tools. Although there are differences how different implementations of object-orientation handle some aspects of the class hierarchy (e.g. repeated inheritance) it is a common characteristic that it is possible to algorithmically navigate the class hierarchy in the same way as it is possible to navigate "isa"-hierarchies of semantic nets. Thus, given a class, it is possible to automatically identify subclasses, and infer superclasses and the characteristics inherited from them.

The formalization of the object hierarchy is usually realized based on the idea that active objects hold references to contained objects and objects they interact with, by which they can access their state and invoke operations. Different implementation technologies/languages might provide different means to realize object hierarchies and dependencies. However it is a defining feature of object-orientation that relationships might be defined between classes at any level of abstraction so that relationships might require specific classes to take part (typification), where the actual class of an object might not be known at the time of specification, in case a superclass is required (polymorphism, i.e. *Contains* and *Partner* are links defined between an abstract class (*Superclass1*) whereas the instantiation of these links is only possible on instantiations of concrete subclasses (*Subclass2* and *Subclass3*)). Further, this allows the specification of behavior (algorithms) for all subclasses and corresponding objects, which is a fundamental abstraction mechanism.

Classes are thought of having a well defined interface[26] that formally defines the accessible constants, variables and operations of an object, while hiding implementation details.

[25] Indeed a variety of aggregate possibilities exist to relate modules, depending on specific implementation of object-orientation (e.g. import and merge in UML). Typically the possibilities to relate classes within a package can be used to relate classes between packages, since packages are basically a way to provide unique class identifiers (namespaces).

[26] Although typically there exist ways to define interfaces and corresponding operations of classes independently, the attributes and operations of classes are perceived as an interface in this thesis for the purpose of simplicity.

However, the specification of the semantics of operations is dependent on implementation technology. In general, any way of specifying of the operations of a finite state machine can be used. Chapter 3 elaborates on the definition of semantics of programming languages that are typically used to define the semantics of operations.

In conclusion, object-orientation integrates an explicitly decompositional view of a software system (object hierarchy) with an explicit representation of abstractions (class hierarchy). Besides maintainability, hierarchical decomposition facilitates the specification of abstractions that are thought to mediate between the implementation technology and the problem domain with *key abstractions* representing the vocabulary of the problem domain and other abstractions representing design decisions (or patterns) against the background of implementation technology (hardware, software, Booch (2004)). Object-orientation is closely related to the development of software engineering tools and respective modeling and programming languages that inherently support object orientation by implementing the object model (Booch (2004), see chapter 3.2.1). Thus, object-orientation combines features of GST with software engineering.

2.2.4 Specific Characteristics of EMS

In general, environmental systems are systems that are addressed within environmental science and management (see chapter 2.1). This section highlights some characterizing properties of environmental systems and practical aspects of related simulation models.

Environmental Systems

According to the broad definition of environment in chapter 2.1 this includes systems related to different scientific disciplines at different spatial and temporal scales and hierarchically composed systems (White et al., 1984). Thus, there is a broad range of systems used in environmental science. A distinguishing feature however is the presence of the following characteristics:

- a relatively high degree of uncertainty,
- a relatively high degree of complexity and
- an explicit geo-spatial reference.

The classification of systems in Karplus (1977) provides the background for clarification of these characteristics that have profound influence on EMS and tool support for EMS. Systems, and respective mathematical models (Dynamical Systems), are arranged along of spectrum from "white-box" to "black-box" (see Figure 2.13).

At the "white-box" end of the spectrum there are fields of investigation that may operate with systems where there is sufficient degree of knowledge and observations[27](e.g. electrical engineering). Models are perceived to reflect the relevant structural and behavioral characteristics of the system with a high degree of credibility, due to reliance of accepted natural laws a system is governed by. At the "black-box" end of the spectrum there is a high degree of uncertainty, due to absence of adequate observations and unavailability of widely accepted governing laws (e.g. when there is human influence, Karplus (1977)). In general, environmental systems can be found along the whole spectrum. However, due

[27]The sufficiency of knowledge and observations is indeed dependent on the objectives.

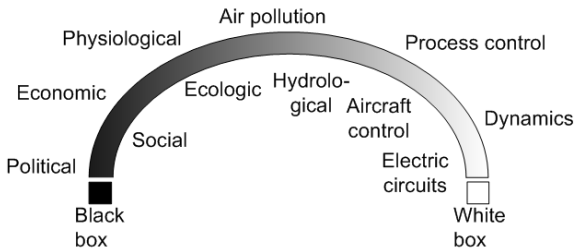

Figure 2.13: The spectrum of systems (Karplus, 1977).

to uncertainty and complexity environmental systems are typically located towards the "black-box" end.

The notions of "uncertainty" and "complexity" require further clarification: Although the term "uncertainty" is defined differently in different contexts, it is possible to identify a common denominator for environmental modeling (Walker et al., 2003; Refsgaard et al., 2007; Brugnach et al., 2008).

> *Uncertainty is any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system (Walker et al., 2003).*

In other words "uncertainty" refers to the degree to which one must expect that an assumed characteristic (e.g. an observed or predicted temperature) deviates from the true characteristic. Walker et al. (2003) characterizes uncertainty in EMS along three dimensions:

- *Location*: uncertainty with respect to system boundary, system structure and parameters, simulator correctness, input data, aggregate outcome uncertainty.
- *Level*: in general uncertainty ranges from the certain to total ignorance, which refers to the fact that it is even unknown that uncertainty exists. Between the extremes it might be possible to assign probabilities statistically, agree on plausible scenarios or at least be aware of the presence of uncertainty.
- *Nature*: there exists *epistemic uncertainty* due to imperfect knowledge (e.g. inaccurate or incomplete data, lack of knowledge) or *variability uncertainty* that is due to assumed inherent randomness of the system.

"Complexity", besides hierarchical composition, commonly refers to a combination of some typical perceptions of environmental systems that may apply to varying degrees in specific cases (see Manson (2001); Vries (2009) for an overview). First, perceptions refer to the general methodological characteristics of studies:

- The system as a whole is not accessible for experimentation or observation of explaining mechanisms (interrelations).
- There is no direct correspondence of existing laws of nature and the variables of the Dynamical System.
- It is difficult to describe complex systems mathematically in terms of Dynamical Systems.

Second, there are behavioral characteristics of complex systems and respective Dynamical Systems:

- non-linear, possibly chaotic, behavior and
- emergent, adaptive and self-organizing behavior.

Some structural perceptions of systems are typically associated with these behavioral characteristics of complex systems:

- Great numbers of system elements interact, with
  - locality of interactions of system elements and
  - feedback relationships between system elements.
- System elements are endowed with memory and anticipation that allow to explicitly trace the evolution in the past and anticipate the future evolution to a certain degree.
- The system is open in that there is input and output (e.g. matter, energy, information).

Although a variety of environmental systems are conceptualized without explicit relation to geo-space, the growing availability of spatially explicit observations at relevant spatial scale, e.g. based on remote sensing, motivates spatially explicit conceptualization of systems (Benenson and Torrens, 2006a). As availability of spatially explicit observations increases, there is the need to adapt conceptualizations of systems and incorporate explicit geo-spatial reference. With explicit geo-spatial reference, issues of uncertainty and complexity may rise, since the number of system elements and their interaction may rise with the introduction of spatially explicit system elements, in contrast to non-spatial models. Further, with reliance on geo-spatial data, issues of uncertain data influence the conceptualization of environmental systems.

The characterization of environmental systems shows that the typical properties of environmental systems are basically associated with two aspects: first, the commonly perceived structural and behavioral characteristics of environmental systems and second, the methodological and epistemological background of related studies. Depending on the goal system-theoretic studies, a single environmental system might be attributed to one field of investigation located along the spectrum, to which a particular scientific discipline or problem domain might be associated. However, according to the interdisciplinary approach, environmental systems might be composed of several subsystems, with different levels of knowledge, varying quality of observations, different scale and several scientific disciplines associated with[28]. The issue of ensuring credibility is particularly relevant and demanding in this context and subject to the design of methods and practice of M&S (see Chapter 2.4).

Simulation Models of Environmental Systems

The interdisciplinary setting of EMS for science and management leads to a typical general development and usage patterns of simulation models, as illustrated in Argent (2004): Successful environmental models, and respective implementations, may exist at four levels during model development and application (see Figure 2.14.

[28] White et al. (1984) refers to the discipline-specific analysis of subsystems as "analysis" and the consideration of issues related to composed systems as "synthesis".

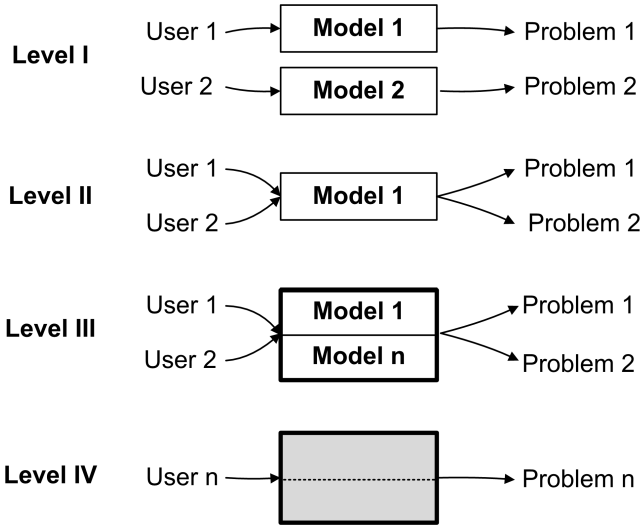

Figure 2.14: Model integration at development and application Levels I - IV (from Argent (2004)) .

At the first level (*Level I*), a scientific model is developed for answering a specific research question for a particular system. The people concerned with system identification and modeling are mainly developers and colleagues in the same or closely related field (Argent, 2004). At the second level (*Level II*), some general utility of a model is stated by application of the model to different research questions and systems, including refinements and model applications. Such models often exemplify a specific conceptual approach and may be used for educational purpose. At *Level III*, a model is applied to a wider range of problems and systems, based on a sufficient number of case studies and the fact that the model describes a process at a sufficient level of detail, " [...] with manageable data requirements (Argent, 2004)." Moreover, the model is combined with other models that represent other aspects of the environmental system. In addition to the original group of users, it additionally encompasses managers that often combine different models for planning analysis. The range of applications may be widened. Communication is often realized by manual, and the like, focusing on data requirements. At *Level IV* the model is used in planning and policy analysis. Code and application are separated and models are used as black-box models. Communication is largely about comparison of data generated by models (Argent, 2004).

In general, the sensibility of the reuse of models, thus the application of a models to different systems and problems requires adequacy in that a model is appropriate to the objectives of a study and that reuse is technically feasible. Although both requirements are fundamental for any reuse of models, the needs of scientific research are perceived to dominate the first levels of model development and usage, technical properties related to reuse gain importance at later levels. The direct reuse of implementations of simulation models within the lifecycle setting has been a major motivation for recent tool development in particular with respect to technical issues (see Chapters 3.3 and 3.4). However, the reuse of models, in particular at later stages, is also based on the generality and the credibility a model gained at early stages of development. Chapter 2.4 elaborates characteristics and requirements of the scientific aspect of EMS, whereas Chapter 3.3 discusses the characteristics of existing tools for EMS including technical reusability.

2.3 Classification and Conceptualization of Dynamical Systems

This chapter presents basic classes of Dynamical Systems and paradigms for the conceptualization of systems. Besides the clarification of basic concepts it is one aim to illustrate common relationships between paradigms, that provide the background for the design of modeling tools since paradigms come along a mathematically founded semantics, but also for reasoning since paradigms are the conceptual framework into which real world perceptions are translated into.

The classes of long-term behavior of Dynamical Systems presented in Chapter 2.2.2 rather refer to observed characteristics rather than characteristics that guide the conceptualization of systems. However some perceived structural characteristics of Dynamical Systems that can be loosely associated with a specific type of long-term behavior (e.g. feedback and non-linearity, see Chapter 2.2.4). In contrast the following classifications rather refer to characteristics of models that modelers necessarily choose before conceptualizing a Dynamical System: the general relation of time and state (Chapter 2.3.1) and the modeling paradigm used for model conceptualization (Chapter 2.3.2).

2.3.1 Basic Classes of Dynamical Systems

Common classifications of Dynamical Systems differentiate between discrete and continuous Dynamical Systems with respect to the evolution of time, state and the specification of corresponding state changes at the first place (e.g. Zeigler et al. (2000)). Figure 2.15 illustrates such classification.

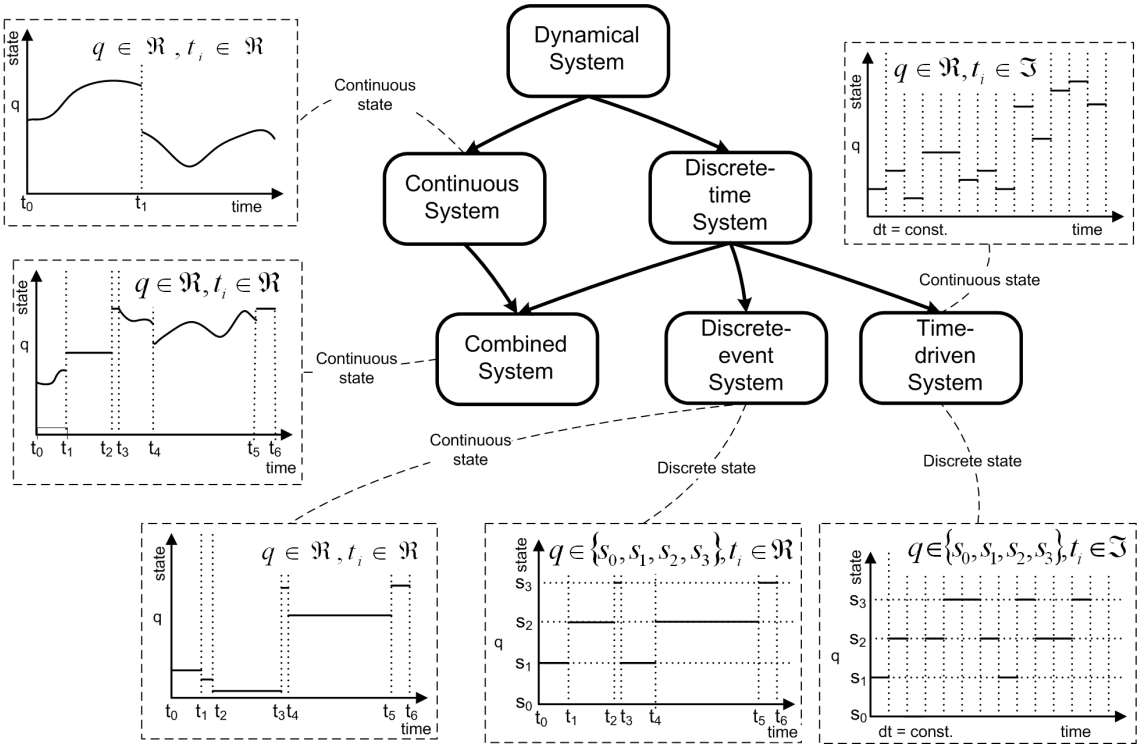

Figure 2.15: Common classification of Dynamical Systems as used in M&S with exemplary trajectories that are composed of piecewise continuous or constant segments.

A *continuous Dynamical System* is thought of changing its state continuously arbitrarily often in any relevant time segment ($Q = \times_{i=1}^{n} \mathbb{R}$ where n is the number of state variables, $T = \mathbb{R}$, $X \in \mathbb{R}$ and $Y \in \mathbb{R}$), with necessarily continuous state. The trajectory is the concatenation of piecewise continuous segments that are defined by a piecewise continuous transition function. The definition of continuous systems is typically associated with the transition function δ given in terms of rate of change ($dq/dt = f(q(t), x(t), t)$, e.g. by means of differential equations). The state-trajectory diagram (dashed line) illustrates the evolution of state with time[29].

In contrast, a *discrete-time system* is thought of changing its state discontinuously a finite number of times in any given time span, with a possibly finite number of possible discrete states ($\forall q_i \in \Omega : q_i \in \mathbb{R} \vee q_i \in \mathbb{I}$, $\forall x_i \in X : x_i \in \mathbb{R} \vee x_i \in \mathbb{I}$, and $\forall y_i \in Y : y_i \in \mathbb{R} \vee y_i \in \mathbb{I}$). The state transition functions of discrete time systems take the general form of maps (e.g. $q(t_{n+1}) = f(q(t_n), x(t_{n+1}), t_{n+1})$), where the application of the transition function defines the value of the next constant segment that is part of the piecewise constant trajectory. Discrete-time systems fall into one of two classes: time-driven systems and discrete-event systems. *Time-driven systems* (or time-stepped systems) change their state at discrete predefined equidistant points in time ($T \in \mathbb{I}$ or isomorphic). Time-driven systems are commonly associated with Difference Equations and automata-based modeling (see Chapter 2.3.2).

In contrast, *discrete-event systems* change state at a finite number of arbitrary times as a consequence of other events, states or time passed ($T \in \mathbb{R}$), thus the times that define segments are not prescribed in advance, but result from application of the state transition function. The evolution of time may modeled by setting Δt explicitly or by the denotation of specific states as conditions that trigger the execution of the transition function. The times of execution of transition functions are typically referred to as "events". Discrete-event systems are associated with discrete-event modeling (see Chapter 2.3.2). The state of discrete-time Dynamical Systems might be discrete or continuous.

Combined systems (also often referred to as *hybrid systems*) combine continuous and discrete-time - typically discrete-event - modeling. The continuous behavior of models is interrupted at times of events that may cause discontinuous state changes. Between events, the system's state might evolve continuously or remain constant. The state transition is a combination of discrete transition functions and rate of change functions, where a condition indicates which transition function is to be applied for a segment. Much of combined modeling has been developed in context of the development of paradigms and respective tools such as hybrid automata and combined continuous and discrete-event modeling (Zeigler et al. (2000); Mosterman and Vangheluwe (2004); Man et al. (2010), see Chapter 2.3.2).

A further basic distinction is made between *deterministic systems* and *non-deterministic systems*. The behavior of deterministic Dynamical Systems is completely predetermined by the initial state, whereas a non-deterministic dynamical system may produce different behaviors under the same conditions. However, determinism does not imply that behavior is completely predictable from initial state only (e.g. chaotic systems). Non-determinism is usually modeled by means of probabilistic elements, where, in contrast to deterministic dynamical systems, several behaviors are possible at a given state with probabilities associated with the different possible behaviors. Figure 2.16 illustrates how non-determinism

[29] In this thesis, 'time' refers to the time in the model ($t \in T$). Wall-clock time or time in terms of computing cycles is denoted as such in this thesis.

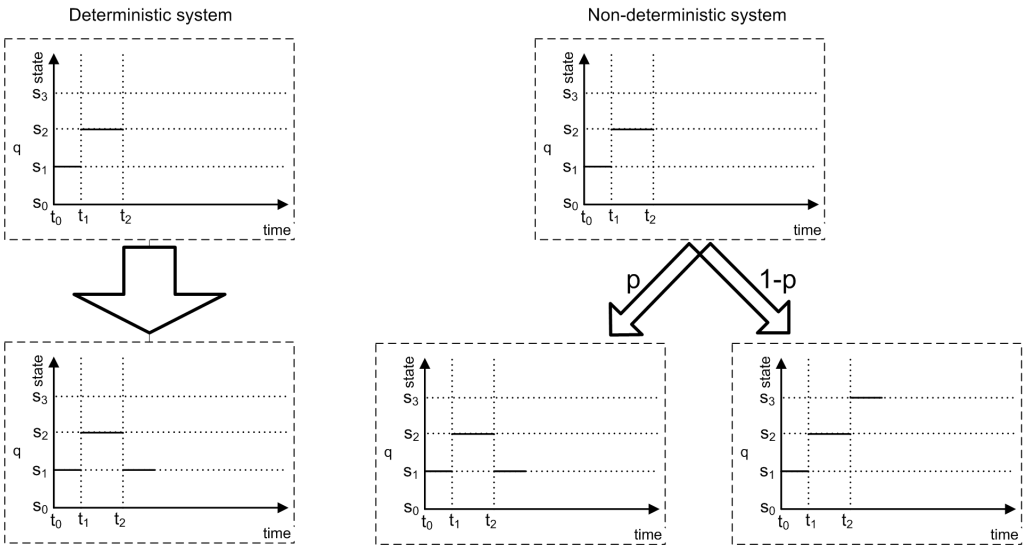

Figure 2.16: Deterministic system (left) is conceptualized such that from each state a particular next state follows, whereas several next states may be possible with given probabilities (*p* and *1-p*) in non-deterministic systems (right).

might be conceptualized. The concrete specification of non-determinism is subject to the specific way (paradigm/language) of model specification, however the use of a great number of random numbers, typically generated by pseudo-random number generators, is a basic feature of non-deterministic simulation models (i.e. a pseudo-random number is computed and compared to probability p in Figure 2.16 in order to determine the next state of a Dynamical System).

In M&S studies, trajectories are usually derived from mathematical models that prescribe structural and behavioral characteristics in a way that allows the derivation of dynamical behavior as state trajectories. The specification of simulation models requires the use of modeling languages that typically follow a particular modeling paradigm that enforce a particular mindset for modeling. There is a variety of available modeling paradigms with different characteristics. Some influential modeling paradigms are shortly characterized in the following Chapter.

2.3.2 Modeling Paradigms for Dynamical Systems

Modeling paradigms provide thought patterns as a framework for the conceptualization of models. According to Villa (2001) *modeling paradigms* are " [...] systems of metaphors that natural system modelers employ to conceptualize the world; they provide semantic frameworks to translate problems into, and allow investigation and simulation of systems by exploiting powerful, inspiring analogies." As such, they help modelers to " [...] conceive, formulate and solve problems by providing semantic structures to organize their view of a system or process (Villa, 2001)." Generally, modeling paradigms are perceived as the conceptual base of tools and modeling languages used for the specification of models (see Chapter 3.3) and as such, they set the framework for both, technical-procedural and the cognitive-epistemological aspects of M&S studies.

Besides the cognitive and epistemic aspect of modeling paradigms (see Chapter 2.4.4),

the usage of paradigms may allow the development and application of paradigm-specific methods of investigation (see examples of difference and differential equations below). Zeigler et al. (2000) shows how a rigorous mathematical definition of modeling paradigms supports the clarification of concepts and the verification of simulators as a prerequisite for building reliable modeling tools for credible modeling processes in M&S. Since the choice of a modeling paradigm typically implies a class of Dynamical System, modeling paradigms are closely related to the basic classes of Dynamical Systems (Chapter 2.3.1), in particular differential equations and difference equations that built the framework for the characterization of continuous and time-driven systems and associated methods of investigation (see below).

Chapter 2.2.1 presented "General Systems Theory" as an abstract paradigm and Chapters 2.2.2 and 2.2.3 Dynamical Systems and object-orientation as mathematical concretizations of it. Further relevant modeling paradigms are presented in the following Chapter with respect to basic conceptual and technical characteristics that can later be set into the technical and cognitive context of system-theoretic simulation studies and MDE.

Differential Equations

Differential Equations are a basic paradigm for modeling continuous Dynamical Systems. A Differential Equation defines the rate of change of a dependent variable as a consequence of the variation of independent variables. An *Ordinary Differential Equation* (ODE) describes the rate of change according to one independent variable, a *Partial Differential Equation* (PDE) according to several independent variables. The following equation presents a (explicit) form of ODE:

$$\frac{d^n y(x)}{dx^n} = F(x, y(x), \frac{dy(x)}{dx}, \frac{d^2 y(x)}{dx^2}, ..., \frac{d^{n-1} y(x)}{dx^{n-1}})$$

, where F is a function of one independent variable x (typically time t) and the value of the dependent variable $y(x)$ and its derivatives $\frac{d^i y(x)}{dx^i}$ at x. Superscripts ($1, 2, \ldots, n$) denote the number of subsequent differentiations at x, where the highest value (n) defines the *order* of the differential equation. The function that makes the differential equation become true is referred to as the *solution* of the differential equation, which may be given at several levels of generality (e.g. set of trajectories or a specific trajectory).

Several coupled differential equations form a *system of differential equations* Given input and output variables and let $q_1, q_2, ...q_n$ be the set of state variables, $x_1, x_2, ...x_m$ the set of input variables and $y_1, y_2, ...y_k$ the set of output variables then, and t the independent

variable, a corresponding system of first order ODE may take the form[30]:

$$\begin{aligned} dq_1(t)/dt &= f_1(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\ dq_2(t)/dt &= f_2(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\ &... \\ dq_n(t)/dt &= f_n(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \end{aligned}$$

$$\begin{aligned} y_1(t) &= f_1(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t)) \\ y_2(t) &= f_2(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t)) \\ &... \\ y_k(t) &= f_k(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t)) \end{aligned}$$

A system of Differential-Algebraic Equations (DAE) adds additional constraints on dependent variables by means of algebraic equations:

$$\begin{aligned} q_1(t) &= f_1(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\ q_2(t) &= f_2(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\ &... \\ q_n(t) &= f_n(t, q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \end{aligned}$$

Partial Differential Equations (PDE) describe continuous systems where the evolution is a function $u(\underline{x})$ of several independent variables $\underline{x} = (x_1, x_2, \ldots, x_n)$ and partial derivatives (denoted by $\frac{\partial u}{\partial x}$):

$$F(x_1, \ldots, x_n, u(x_1, \ldots, x_n), \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1 \partial x_1} \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_n}, ...) = 0$$

For both, ODE and PDE a basic distinction is made between *linear* and *nonlinear* (systems of) differential equations. Linear Dynamical Systems are generally associated with the changes of output being proportional to changes of input, whereas the reaction of non-linear systems is not proportional. More specifically, linear (systems of) differential equations are defined by the unknown function under consideration (e.g. $y(x)$, $u(\underline{x})$) and its (partial) derivatives (e.g. $\frac{dy(x)}{dx}$, $\frac{d^n y(x)}{dx^n}$, $\frac{\partial u}{\partial x_1}$) are only combined linearly (e.g. no multiplications, no powers $\neq 1$)[31]. Otherwise systems of differential equations are *non-linear*. A comprehensive discussion of methods is beyond the scope of this thesis, however, it is a basic aspect to notice that there appears to be a relatively complete theory that provides means for finding solutions and characterization of long-term behavior for linear systems, whereas non-linear systems appear to be less amenable for investigation in particular when finding solutions[32] (Boccara (2004), see Chapter 4). However, the range of possible be-

[30] Higher-order differential equations can be transformed into a system of first-order differential equations

[31] Formally a single linear differential equation can be written in the form $\frac{dy_i^n}{dx^n} + a_{n-1}\frac{dy_i^{n-1}}{dx^{n-1}} + a_{n-2}\frac{dy_i^{n-2}}{dx^{n-2}} + ... + a_1\frac{dy_i}{dx} + a_0 y = f$, where a_i and f might be functions that are not dependent on y.

[32] However, linearization is a common technique that facilitates the application of methods for linear

haviors of nonlinear systems is relatively great compared to linear systems in that linear systems typically exhibit fixed point or diverging long-term behavior, whereas non-linear systems may exhibit "more complex" periodic (limit cycle) and chaotic behavior as often observed at natural real systems (e.g. strange attractors, see Boccara (2004)).

An exemplary ecological model specified by nonlinear first-order ODE is the Lotka-Volterra predator-prey model, where q_1 denotes the size of a prey population and q_2 denotes the size of predator population that feeds on the prey (a is the growth rate of prey without predators, b predation rate of predators, c qrowth rate of predators per prey, d death rate of predators without feed):

$$dq_1(t)/dt = aq_1 - bq_1q_2$$
$$dq_2(t)/dt = cq_1q_2 - dq_2$$

Depending on values of parameters a, b, c, d and initial state, in the long run either both or one species die out, or there is an oscillation of number of individuals of the two species (limit cycle, see Boccara (2004) for discussion).

Spatio-temporal variability of systems can be explicitly modeled by means of PDE, by setting time and space coordinates as independent variables. The heat equation that models the spatio-temporal distribution of heat in a region exemplifies spatio-temporal modeling with PDE:

$$\frac{\partial q}{\partial t} - \alpha \left(\frac{\partial^2 q}{\partial c_x^2} + \frac{\partial^2 q}{\partial c_y^2} + \frac{\partial^2 q}{\partial c_z^2} \right) = 0$$

, where c_x, c_y and c_z are space coordinates, t time and q the temperature[33].

A major goal of system-theoretic investigations is the identification of correspondences between the perceived structure of real systems and properties of Dynamical Systems. At a general level, a primary source for complex behavior is attributed to feedback (Zeigler et al., 2000) that shows by a state variable influencing its derivative directly or indirectly via influencing state variables that are influential to the derivative (for example Lotka-Volterra). Generally, positive feedback loops (positive influence on derivative) tend to promote indefinite growth (unstable), whereas negative feedback loops (negative influence on derivative) stabilize a system towards stable limit sets (e.g. fixed points, limit cycles). Both are typically found in a variety of systems. Whereas linear negative feedback is rather associated with fixed points limit sets, non-linear feedback may enable limit cycles or chaotic behaviors (see Boccara (2004) or Zeigler et al. (2000)). A Dynamical System defined by differential equations with a strange attractor must have at least three state variables (Poincare-Bendixson theorem) and typically at least one non-linear term (the Lorenz system has three state variables and two nonlinear terms). However, although some general patterns have been identified, in the empirical sciences it is typically not possible to combine usable models from "building blocks" (e.g. feedback loops) based on given perceptions, but model building in non-trivial situations with a certain degree of uncertainty is rather embedded within exploratory experimentation processes (see Chapter

systems to non-linear systems (e.g. stability analysis (eigenanalysis), see Boccara (2004))

[33] This equation reduces to $\frac{dq}{dt} - \alpha \left(\frac{d^2 q}{dc_x^2} + \frac{d^2 q}{dc_y^2} \right) = 0$ in two-dimensional case or $\frac{dq}{dt} - \alpha \left(\frac{d^2 q}{dc_x^2} \right) = 0$ in the one-dimensional case

2.4).

A fundamental ingredient of experimentation processes is the identification of solutions. In general, there are two ways of finding solutions of differential equations: analytical and numerical. Whereas analytical solutions typically provide exact solutions at some level of generality[34], numerical solutions are based on step-by-step execution of approximative algorithms, where each execution provides a specific solution to a specific initial value problem in form of a corresponding trajectory. In the context of M&S, systems are typically assumed to retract from analytical investigation, such that numerical treatment is required.

The exactness of any numerical solution is subject to the adequacy of approximation, which depends on properties of the system (e.g. stiffness, sensitivity), the solution algorithm and the length of the trajectory. In general, exact analytical solutions are to be preferred to numerical, however analytical solutions are typically not available for complex systems, in particular those described by non-linear differential equations (Boccara, 2004). Therefore, a variety of numerical algorithms exist for solving initial value problems for systems of differential equations that are typically built upon discretization and linearization, where calculations are made iteratively for discrete time intervals based on a linearized version of the system of nonlinear differential equations at specific values. However, there are no general-purpose numerical algorithms such that at the most general level, algorithms differ with respect to type (PDE, ODE, DAE) and order of differential equations[35]. A variety of algorithms at different levels of generality (see Galassi et al. (2011), Eaton et al. (2011) or MAT (2012)) taylored towards specific types of Dynamical Systems and related issues. Each algorithm represents a trade-off between generality, stability, accuracy and performance. In general, knowledge about properties of Dynamical System helps choosing a adequate algorithm.

In practice, some relevant properties are directly visible from given differential equation (e.g. order, type (ODE or PDE)), whereas other properties (e.g. stiffness) require analysis, for which (systems of) nonlinear differential equations may be amenable for linearization (e.g. Jacobian) in order to apply theory of linear differential equations (e.g. use eigenvalues of Jacobian to characterize stiffness). For PDE, partial derivatives can be approximated by means of discretization, e.g. a (system of) PDE is transformed manually into a system of ODE (e.g. method of lines) and this is solved by means of a conventional ODE solver (e.g. Runge-Kutta). Out-of-the-box numerical algorithms take at least the differential equations and initial states as input and apply a discretization scheme automatically (algorithms may require more input such as Jacobian, accuracy, time step, discretization grid etc.). A further common approach to solve initial value problems is to specify the complete approximation scheme explicitly and manually specify the algorithm, e.g. as an discrete-time model in form of difference equations (see below), which gives space for optimization of both accuracy and performance. If relevant characteristics of a model are not clear from analysis, it is possible to apply an explorative experimentation process that entails degrees of freedom with respect to discretization schemes, which indeed requires modelers to be aware of the relationship between the differential equations and the approximation method used.

However, along analytical and numerical methods a variety of relatively simple prototypical Dynamical Systems (e.g. Lotka-Volterra, Lorenz system) with exemplary behavior

[34] Differential equations are further classified according to analytical methods that can be applied, which are not of interest for this thesis (e.g. constant coefficients, homogeneous, autonomous).

[35] Higher-order ODE can easily be transformed to systems of first-order ODE.

have been developed at the base of differential equations, that combine typical observed patterns (i.e. coupled oscillations, chaotic behavior) with structural perceptions (i.e. competing species with limited resources) with a rigorous mathematical framework. The simplicity of exemplary models reflects the generality of underlying assumptions and often refers to properties that may be found at many different systems, which aligns well with the primary goal of General Systems Theory (e.g. Lotka-Volterra equations as a general model of the evolution of competing species, see Boccara (2004) for more examples).

In general, the theory of differential equations is relatively well-developed (Zeigler et al. (2000)) and scientists in the sciences are typically relatively familiar with it (Toffoli, 1984). However, since the roots of development of theory of differential equations date back into times where digital computers were not available, this paradigm appears to be rather aligned with "pencil-and-paper" analytical practice than with the requirements and possibilities of digital computing (e.g. collaborative modeling and discreteness). Moreover, the application of such general models to specific real systems, e.g. for the purpose of quantitative predictions, typically requires refinement, estimation of parameters and the inclusion of complex boundary conditions and adaption to actually observed data. The simplicity of models, from which follows analytical tractability, might not be given in this case, which is perceived as a major drawback in many cases and a major motivation of the use of Cellular Automata as a modeling paradigm (see Chapter 4 for further elaboration).

The theory of differential equations is closely related to the theory of difference equations as described below.

Difference Equations

The *Difference Equation* paradigm is a paradigm for modeling time-driven models. The evolution of the Dynamical System is described by a system of difference equations defines a series of system states when applied iteratively starting with an initial state (x_0). In a general, difference equation has the form

$$x(n+k) = F(x(n+k-1), x(n+k-2), \ldots, x(n))$$

, where the value of the $(n+k)$th x of the series is a function F of former k values of x[36]. If the terms $x(i)$ in F are combined linearly the difference equation is linear, nonlinear otherwise. Like differential equations, (systems of) difference equations with more that one independent variable are referred to as *partial difference equation*, *ordinary difference equation* otherwise. Partial difference equations typically result from discretization of PDE (see Kelley and Peterson (2001)). Systems of difference equations are classified accordingly.

In time-driven systems one independent variable is time t and each iteration is associated with a constant progress of time Δt. A system of first order ordinary difference equations

[36] k defines the *order* the difference equation.

with time as independent variable and input and output takes the form:

$$
\begin{aligned}
q_1(t+1) &= f_1(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\
q_2(t+1) &= f_2(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t)) \\
&... \\
q_n(t+1) &= f_n(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_m(t))
\end{aligned}
$$

$$
\begin{aligned}
y_1(t) &= f_1(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t)) \\
y_2(t) &= f_2(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t)) \\
&... \\
y_k(t) &= f_k(q_1(t), q_2(t), ..., q_n(t), x_1(t), x_2(t), ..., x_n(t))
\end{aligned}
$$

Like differential equations, properties of difference equations (e.g. solutions, long-term behavior and stability) might be found analytically to varying degree, depending on the type of difference equations[37]. In general, there is relatively strong methodological background for analysis of linear difference equations, whereas nonlinear difference equations might require numerical analysis, in particular to find solutions (Kelley and Peterson, 2001).

Compared to differential equations, the derivation of state trajectories by means of simulation is straightforward by iterative application of the difference equations. However, numerical issues may arise depending on characteristics of the model (e.g. sensitivity), objectives of the study and limited word length and precision of numbers. Thus, as is the case with differential equations, the evaluation of correctness might require exploration of the relationship of model and simulator by means of experimentation.

In general, difference equations show qualitatively similar behavior to differential equations, when abstracting from discreteness. The approximation of differential equations by means of difference equations is common technique approach to investigate differential equations. Nonlinear relationships and feedback can be modeled analogously to differential equations. However, relatively simple non-linear discrete systems may exhibit chaotic behavior, such as the logistic map with one state variable and one non-linear term (see 2.2.2).

The following equations show a difference equation Lotka-Volterra model derived by application of Euler integration scheme to the ODE version, where superscript t denotes the value for time t and Δt the size of the time step:

$$
\begin{aligned}
q_1(t+1) &= q_1(t) + (aq_1(t) - bq_1(t)q_2(t))\Delta t \\
q_2(t+1) &= q_2(t) + (cq_1(t)q_2(t) - dq_2(t))\Delta t
\end{aligned}
$$

Spatial modeling might be realized by a discretization of space, where each spatial unit is modeled by means of variables and spatial relationships are described by the difference equations. The following illustrating difference equations are a discretization of

[37] There is a great degree of overlap with differential equations, since differential equations might be perceived as the limit case, where the difference of independent variables (e.g. Δt) goes to zero

one-dimensional heat equation following a Forward Time Centered Space discretization scheme (from Recktenwald (2012)):

$$\begin{aligned} q_1(t+1) &= q_1(t) \\ q_i(t+1) &= rq_{i+1}(t) + (1-2r)q_i(t) + rq_{i-1}(t) \\ &\dots \\ q_i(t+1) &= rq_{i+1}(t) + (1-2r)q_i(t) + rq_{i-1}(t) \\ q_n(t+1) &= q_n(t) \end{aligned}$$

Here, $r = \alpha\Delta t/\Delta x^2$. The subscript i refers to the position of the inner spatial elements ranging from 2 to $n-1$, thus space is referred to via vector index. 1 and n refer to the first and last spatial element at the boundary[38]

In general, difference equations are widely used in M&S, since they combine the possibility to represent a variety of interesting behaviors with relatively straightforward digital simulation and a relatively well-developed theoretical background. Although the discreteness of Difference Equations may reflect properties of the modeled real systems, e.g the logistic map models the growth of a population that is made up of discrete individuals that (N) with limited resources (inhibited growth model), Difference Equations are often simply used as a discrete approximation of Differential Equations for analysis and simulation (such as the examples above). In this case however, (Winsberg, 1999) describes that discretization often influences the conceptualization of models in that assumptions are adjusted according to the discretization scheme, thus modelers synthesize properties of discretization schemes into models and the discretization scheme becomes (part of) the used modeling paradigm.

As with differential equations much of development of difference equations took place irrespective of developments of digital computing, thus specific possibilities and requirements (e.g. collaborative modeling and partial specification of models) are not well developed. However, a variety of prototypical models (e.g. Logistic map) have been developed, but, like differential equations, the application to real systems typically requires adaption and rather leads to numerical treatment of models in practice.

Stocks-and-Flows

With the *Stocks-and-Flows* paradigm the system is conceptualized by means of stocks that accumulate quantities of interest. Quantities vary by means of flows that continuously transport quantities between stocks or into and out of the system. Flows are dependent on the state of the system, which requires the modelers to specify information flows in the system following the idea of regulation in the sense of cybernetics and control theory.

The Stocks-and-Flows paradigm is perceived to offer a high degree of clarity to environmental modelers in that it is well aligned with the needs to model material flows that are of special interest of environmental modelers (Ford, 2009).

Thus, whereas the state (Q), input and output of the system is given explicitly (stocks, sources and sinks), the transition function (δ) follows from the structure of the model (flows, controls etc.). However, it is a basic characteristic of the Stocks-and-Flows paradigm,

[38] In the example q_1 and q_n are set to a constant value, however these boundary values could be set by time variable input: $q_1^{t+1} = x_1^{t+1}$ and $q_n^{t+1} = x_2^{t+1}$, where x_1 and x_2 denote inputs at the boundary.

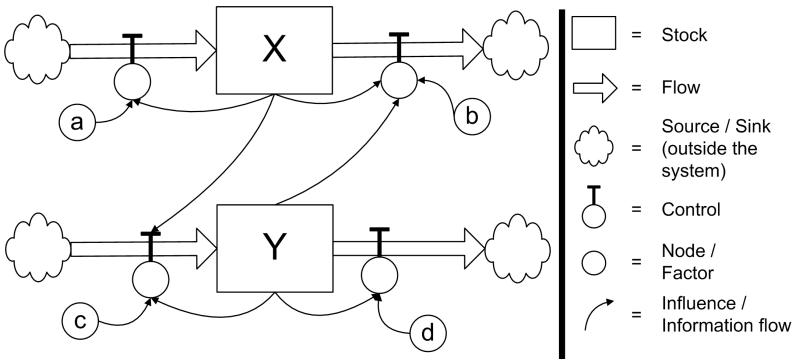

Figure 2.17: Lotka-Volterra model in System Dynamics stocks-and-flow notation.

that its mathematical semantics is given by a translation into Differential (-Algebraic) Equations or respective discrete approximations (e.g. difference equations) for numerical treatment.

Spatial modeling is not explicitly supported by the Stocks-and-Flows paradigm. However, it is particularly tailored to directly express feedback cycles that are perceived to account for non-linear and complex behavior of many environmental systems.

System Dynamics is a popular modeling paradigm in environmental modeling that follows the Stocks-and-Flows paradigm. It is a distinguishing feature of System Dynamics that its use is particularly associated with the aim to support the development and refinement of mental models, thus the support for cognitive processes of simulation studies (Doyle and Ford, 1998). Figure 2.17 presents the specification of the exemplary Lotka-Volterra predator-prey model given above according to a common System Dynamics notation.

Block-based Modeling

With the *Block-based Modeling* paradigm a system is thought of being constructed of blocks that are connected via sending and receiving signals. Blocks create or process incoming signals and may send signals to other blocks. A variety of different types of blocks exists where there are basic blocks that represent basic mathematical operations, such as standard signals (functions), integration, gain, and basic algebraic operations. In addition, there are rather specific ("high-level") types of blocks that allow a high degree of structural similarity of models and systems in particular in the domain of electrical engineering (e.g. for controller design, Borshchev and Filippov (2004)). The state of respective Dynamical Systems Q is basically represented by signals, that can be observed and the transition function δ is specified by the blocks and their connections that process signals. The mathematical semantics of Block-based Models is given by an interpretation as Differential (Algebraic-) Equations (Borshchev and Filippov, 2004).

There exist a number textual and graphical languages and tools that support block-based modeling, that however come along with specific types of blocks and extensions (e.g. Simulink, SimVis, Borshchev and Filippov (2004)).

Figure 2.18 presents the specification of the exemplary Lotka-Volterra predator-prey model given above using Simulink.

Following the "traditional" Block-based Modeling paradigm, the direction of signal flow is prescribed by the modeler, so that causality, which depends on the assumed structure of

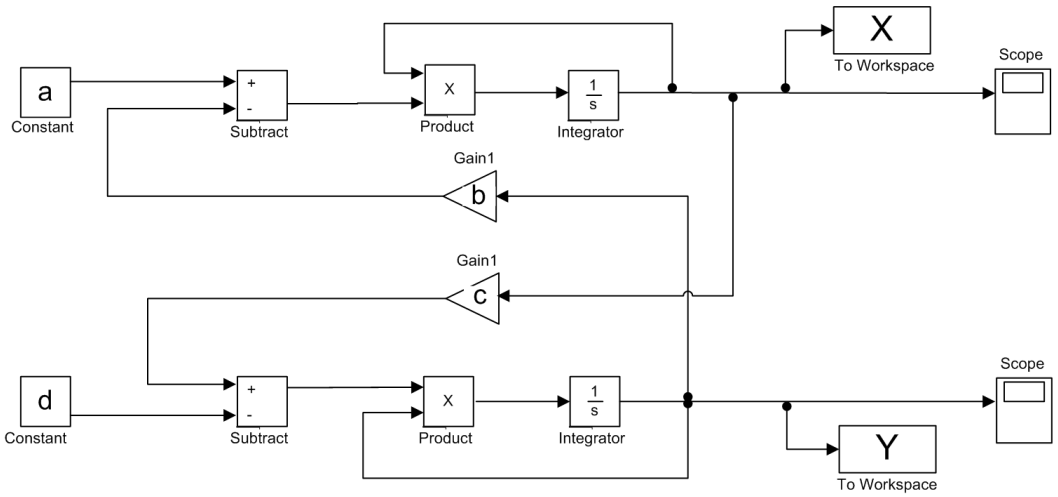

Figure 2.18: Lotka-Volterra predator-prey model in a block-based notation based on a Simulink model published in (Bai and Sander, 2012).

the system and the objectives of the study, is explicitly modeled (see Figure 2.18). Usually, variables are meant to have a "physical" meaning such as "position" or "velocity" (Borshchev and Filippov, 2004). Acausal *Physical Modeling* aims at more general modeling that strives at higher structural similarity of models and physical systems by not explicitly modeling the direction of flows. Causalities might be (automatically) derived when the objectives (i.e. the variables to observe) of a study are given (Modelica, Bond Graphs, Fritzson (2003)). Block-based Modeling can be used for combined modeling where discrete-events are triggered when continuous variables cross predefined thresholds, causing the immediate application of different DAE e.g. (Modelica, VHDL-AMS, Simulink/Stateflow, Mosterman and Vangheluwe (2004); Man et al. (2010)). The simulation of continuous behavior and the detection of state events is typically built upon numerical solvers for differential equations (Mosterman and Vangheluwe, 2004).

Although more powerful than System Dynamics in terms of the equation systems that can be expressed, Block-based Modeling and Physical Modeling have found less acceptance in the environmental modeling community, seemingly due to the its orientation towards engineering applications (Borshchev and Filippov, 2004).

Automata-based Modeling

The basic notion of "automata" as modeling paradigm is that a system is conceptualized as an automaton or a collection of interacting automata. An automaton is an entity that changes its state at discrete points in time by "mechanically" following a rule that depends on the state and input of the automaton[39]. Modeling with automata focuses on the conceptualization of the transitions between states, given (a series of) inputs into the automaton. The set of states (Q) is typically explicitly defined as qualitative attributes (e.g. a set of

[39]In many fields of automata-based modeling the passage of time is not explicitly modeled, but it is the causal structure of state transitions that is of interest. Zeigler et al. (2000) refers to this time as *logical time*. The application of this paradigm in environmental modeling however usually assumes that there is a proportional relationship between the passed time in the model ($t \in T$) and the time passed in the modeled real system.

specific possible values) or by means of numerical variables and compositional modeling is supported by connecting inputs and outputs of automata, where each automaton provides a partial definition of the overall state and transition function. The transition function δ is defined by means of transition rules that might be as simple as a lookup-table. In practice however, rules may be described using computer languages that are limited only by the limits of digital computing (i.e. general-purpose programming languages). Figure 2.19 illustrates different approaches to define transitions in automata-based models.

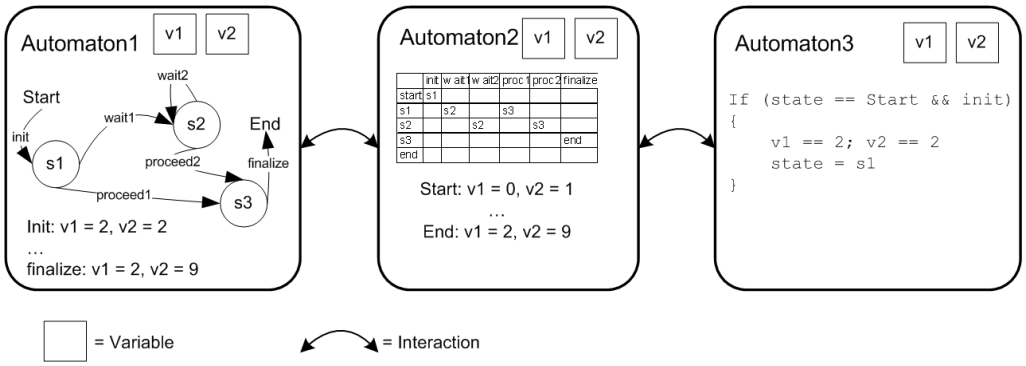

Figure 2.19: Schematic illustration of automata-based modeling, where automata are perceived as transitioning between different defined states.

Basic automata are characterized by discrete time and discrete state and a deterministic rule, so that correctly implemented digital simulators are numerically correct, irrespective of the individual properties of models and objectives of a study. In practice however there is a considerable amount of variation within this paradigm.

Besides the introduction of non-determinism and continuous state, Hybrid Automata appear remarkable in M&S. Hybrid Automata are a canonic representation of combined systems, where in certain states continuous behavior - typically modeled by means of Differential Equations - occurs. States are associated with thresholds and which when crossed cause the transition to another state with possibly different continuous behavior (different equations). The simulation of continuous behavior and the detection of state events is typically built upon numerical solvers for differential equations (Mosterman and Vangheluwe, 2004).

In environmental modeling there are two specializations of automata that attract special attention: Agent-based Modeling and Cellular Automata. In contrast to other modeling paradigms (i.e. Differential Equations, Finite State Machines) there is a great amount of ambiguity in the use of Agent-based Modeling and Cellular Automata, because in practice of environmental modeling, both terms denote a variety of different concepts (see Drogoul et al. (2003) for a discussion of agent-based modeling and Chapter 4 for Cellular Automata). However two important characteristics are commonly associated with Automata-based Modeling: First, the simplicity of models and, second, a local view when describing behavior, while observing global characteristics. Although it is possible to use automata as an approximation for other paradigms (e.g. PDE), automata-based modeling is commonly applied against the background of complex behavior of systems that is assumend to emerge from interactions of the assumed elements of the system, where other well established paradigms, in particular DAE and Difference Equations, have shown to have limited capabilities (Wolfram, 2002; Fredkin, 2003).

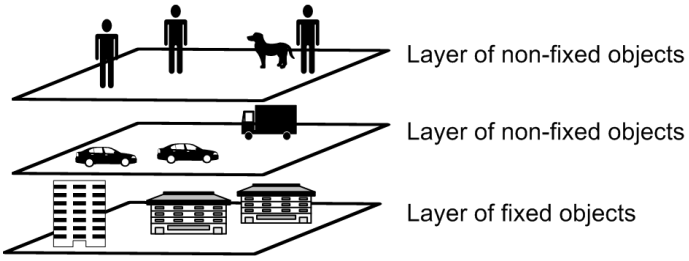

Figure 2.20: Conceptualization of a Geographic Automata System with layers of objects (automata) of different types (from Benenson and Torrens (2006b), modified).

Benenson and Torrens (2006b) suggest the use of *Geographic Automata Systems* (GAS) for spatially explicit modeling with automata, where the system is composed of a number of interacting *Geographic Automata* that either follow an Agent-based or Cellular Automata conceptualization. Figure 2.20 illustrates the notion of a system that is composed of geo-spatially localized interacting objects, where different types of objects are organized in separate layers. Objects are modeled by means of automata. GAS should particularly support research of emergence and self-organization in environmental systems. Geographic Automata have an explicitly modeled geospatial dimension (extent, location), geospatial relations and geospatial movement that can be conceptualized according to respective rules (geo-referencing rules, neighborhood rules, movement rules). The specific form of rules is not given, but it is suggested to implement GAS as object-oriented extensions of Geographic Information Systems (GIS), so that programming languages and GIS provide basic conceptual frameworks for the conceptualization of GAS.

Discrete-event modeling

Discrete-event modeling is characterized by the explicit conceptualization of the passage of time and resources, where the passage of time is modeled by scheduling events at arbitrary times. In discrete-event modeling, there is a focus on relevant parts of systems and points in time, which are those parts that are directly affected by events. Different parts the dynamical system can be explicitly conceptualized as entities that evolve asynchronously, in contrast to time-driven systems. However, the possibility of several different events occurring at the same time possibly affecting shared resources requires explicit modeling of synchronous events (e.g. by explicitly ordering events by priority).

Historically, different world-views emerged from the development of different discrete-event modeling languages: event-scheduling, process-interaction and activity-scanning. These world-views provide different conceptual frameworks, thus paradigms, for the conceptualization of discrete-event models . Although it is state-of-the-art to combine these three approaches, they are characterized separately in the following for the sake of clarity.

In the *event-scheduling* paradigm the dynamics of the discrete-event dynamical system is conceptualized by means of events, as illustrated in Figure 2.21. The state of the system (Q) is modeled by means of a set of variables, that may be described compositionally by means of different system elements where each specifies its own state variables. An event prescribes the change of the system and may schedule future events. Thus, the transition function δ is given by means of the set of events. An event is pre-scheduled for the time of the event (time event) and might be conditioned by the availability of resources. The

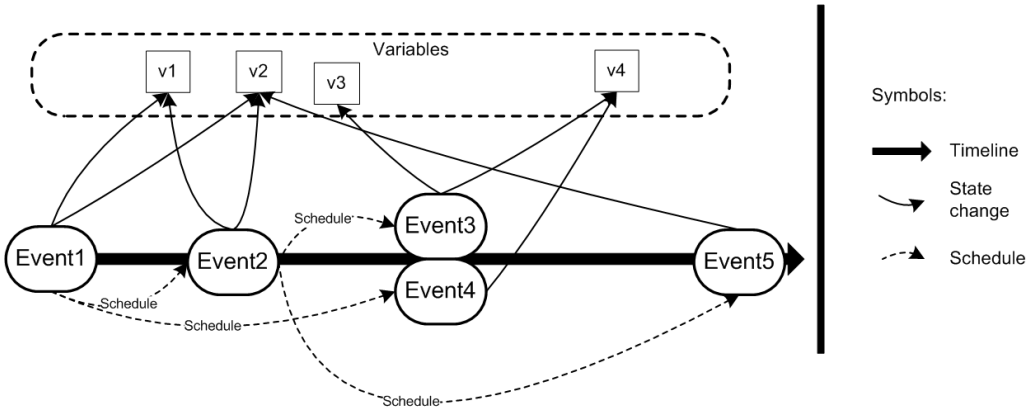

Figure 2.21: Illustrative sketch of the event-scheduling world view of the discrete-event modeling paradigm.

transition function is conceptualized from a global perspective, thus the changes made to all relevant entities are described by and assigned to one event. However, (Zeigler et al., 2000) introduced a method for compositional specification of event-scheduling models. The specific concepts used for modeling state changes is subject to specific modeling languages (e.g. general-purpose programming languages).

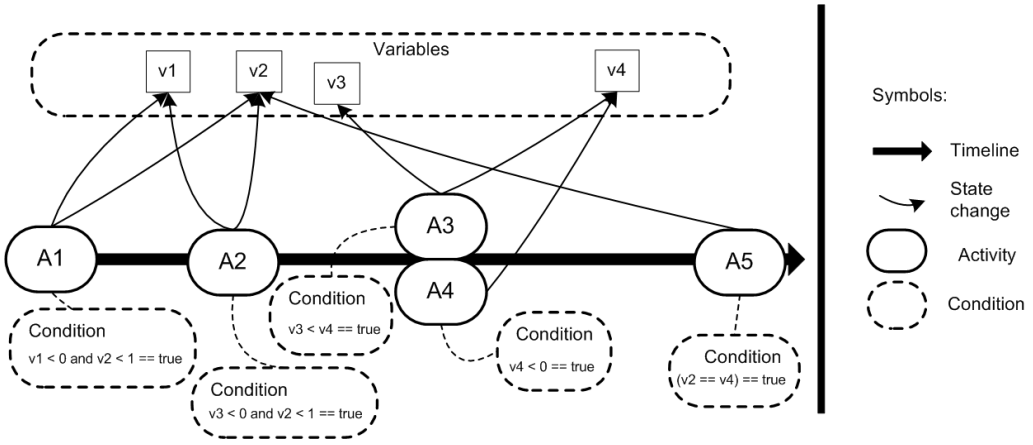

Figure 2.22: Illsutration of the activity scanning world view of discrete-event modeling paradigm.

In *activity scanning* world view *state events* - typically called "activities" - take place when specified conditions are met (see Figure 2.22). Conditions are evaluated at a constant discrete time step, however the *three-phase approach* is a combination of activity-scanning and event scheduling, where events might be scheduled (time events) or conditional (state event). Conditions are evaluated at times of events. Thus, the transition function (δ) is given as set of time and state events.

In the *process interaction* world view the system is thought of being composed of interacting processes that might compete for resources, as illustrated in Figure 2.23. The state of the system is basically defined by variables defined for processes and resources. A process has its own state as part of the whole dynamic system's state. A process undergoes a

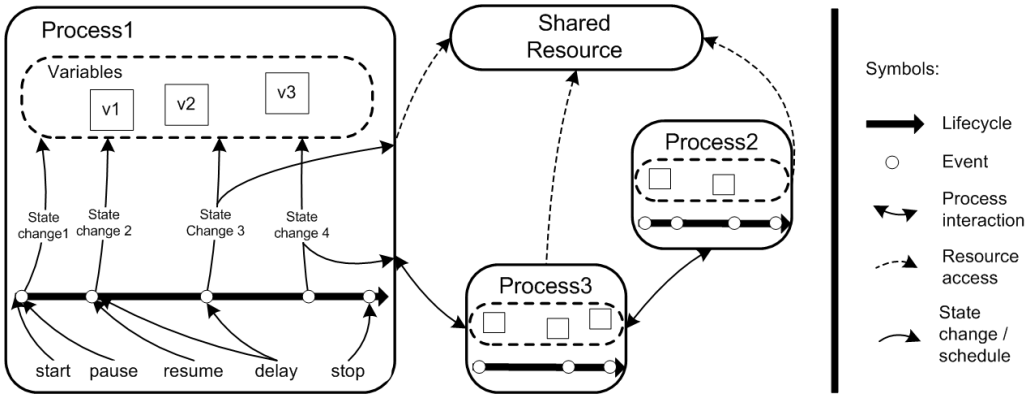

Figure 2.23: Schematic illustration of the process interaction world view discrete-event modeling paradigm.

time ordered sequence of activities referred to as the process-lifecycle where the transition function is defined by the process-lifecycles of all processes. Each process-lifecycle can be started, stopped, resumed, delayed and paused by itself or by other processes. Single activities might change state, or cause the current process to pause/stop and another process to be activated. In contrast to event-scheduling world view, process interaction conceptualizes the system's behavior based on processes, thus a "local" view on the system.

Discrete-event modeling can be combined with continuous modeling where continuous behavior of the system is described by differential equations. At times of events the continuous behavior is is interrupted and the event may change state discontinuously. State events are specified by conditions that may be met when behavior changes continuously, which is typically simulated using numerical integration algorithms (Helsgaun (2001); Fischer and Ahrens (1996) describe combined process-oriented modeling, Zeigler et al. (2000) describes combined event-oriented modeling).

Besides conceptual alignment with systems that are perceived to evolve at arbitrary time-steps, discrete-event modeling is regarded as attractive because it is " [...] intrinsically tuned to the capabilities and limitations of digital computers (Zeigler et al., 2000)." This allows for the relatively efficient implementation of powerful simulators (see Chapter 3.3), in particular when compared to difference equations for specific types of Dynamical Systems and directly supports compositional specification of systems in support of collaborative modeling. However, since discrete-event modeling evolved from the development of languages and tools, there is remarkable conceptual ambiguity in concepts and there does not appear to be a theoretical methodological background compared to differential and difference equations, particularly with respect to analytical methods (see Zeigler et al. (2000), see Chapter 3.3.3).

Spatial modeling: Geographic Information Systems

GIS and its foundational concepts rather provides a paradigm for representing geo-space, not Dynamical Systems, which might be combined with paragdigms for the conceptualization of Dynamical Systems (e.g. Geographic Automata Systems, see Chapter 2.1) The representation of geo-space in context of digital computers by means of geodata is heavily influenced by *Geographical Information Systems* (GIS) and the concepts they are based on.

> *In a narrow sense a GIS is software that is used to collect, manage, analyze and display geodata. In a broader sense, the term GIS in addition subsumes hardware, data and additional applications (Müller, 2001a).*

As such, GIS provide a framework for retrieval and combination of geo-data from different sources (i.e. remote sensing, analog mapping and digitization and GPS measurement) with data management and data analysis and, finally, the provision of application-oriented products based on visualization. This is based on a transformation of geodata into data that conforms to a common GIS-specific data model (see Figure 2.24), which allows for integration and application of GIS-specific services and operations.

> *Geodata is data with a spatial reference that can be used to draw a reference to location on the earth' surface (Müller, 2001b).*

Spatial reference is usually based on 2-dimensional or 3-dimensional coordinates according to a geospatial *coordinate reference system* (Müller, 2001b).

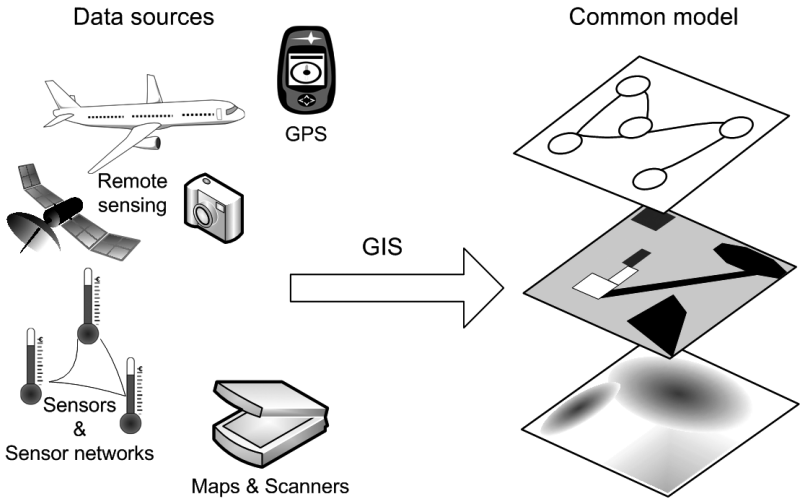

Figure 2.24: GIS provide means to integrate data from different sources.

In general, geospatial features are commonly classified as discrete objects with spatial boundaries or as fields with characteristics that vary continuously or discontinuously in the complete space under consideration. A further commonly used basic abstraction is the network as a set of connected spatial objects. It is a further characteristic feature of GIS that different spatial characteristics are conceptualized by means of overlaying layers, where each layer represents a specific thematic aspect (e.g. vegetation, topography, temperature, communication; see Figure 2.25 (left)).

Figure 2.25 illustrates the basic services that GIS provide (d. By et al., 2000):

- data management and storage with efficient access (e.g. through respective data models (e.g. points, lines, polygons, raster etc.), spatial indexing and meta-data),
- geospatial referencing (e.g. common reference systems, reference system transformations, georefencing, geocoding),
- visualization (e.g. cartographic visualization and animation) and

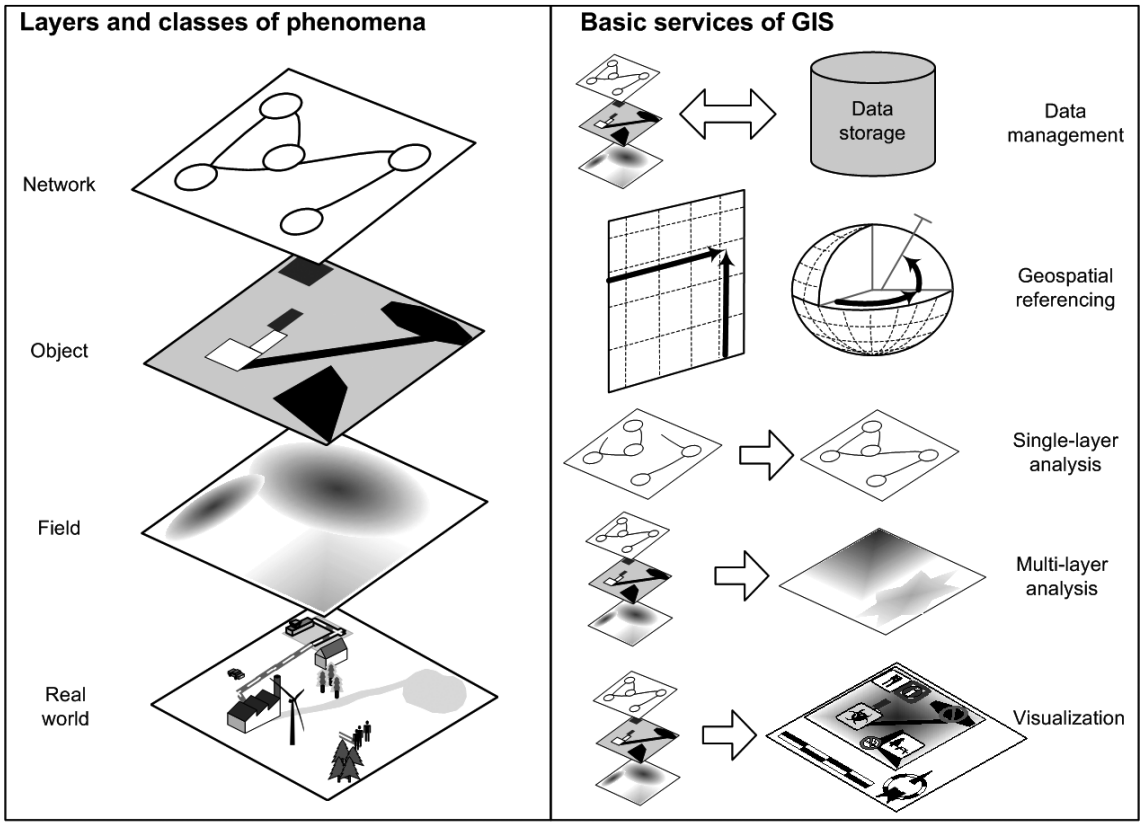

Figure 2.25: Basic concepts if GIS.

- geospatial analysis (e.g. aggregation, error detection).

There are two basic forms of geospatial analysis: *single layer data analysis*, e.g. for retrieval, classification, generalization and measurement of spatial features (e.g. area, distances), and *multiple layer data analysis* (e.g. overlay functions). Typical classes of GIS-based analysis are:

- neighborhood analysis (spatial search, buffering, topological operations, spatial interpolation/extrapolation, spatial filtering),
- topographic analysis (aspect,slope) and
- connectivity analysis (e.g. contiguity, network analysis).

GIS-based processing of geodata typically consists of a series of applications of operations to source data. Time-stepped dynamical systems can be modeled by means of iterative application of operations that model a single transition of the dynamical systems. There exist various ways to describe such iterative applications of operations which depend on specific GIS-tools (see Chapter 3.4.3).

Due to an idiosyncratic development of GIS software, the field of GIS was characterized by a heterogeneity of concepts. However, in order to overcome limitations of interoperability of GIS software, influential standardization took place since the early 90's, with participation of major GIS vendors, GIS users groups and public administration. The Open

Geospatial Consortium (OGC, OGC (2012)) and International Standardization Organization Technical Committee 211 Geographic Information / Geomatics (ISO, ISO (2012)) published a variety of standards and norms which, although indeed not fully implemented by all participants, lead to convergence of core concepts in the field of GIS. These standards are specified in a series of norms and standards that prescribe a data model with operations, which is modeled by means of an object-oriented meta-model: the *General Feature Model*. As a comprehensive discussion of this standardization is beyond the scope of this thesis, it is further discussed in relation to the topic of this thesis in Chapter 5.3.

2.3.3 Conclusion: Relating Modeling Paradigms

Given the notion of modeling paradigm as a conceptual framework for the conceptualization of systems, a number of presented notions qualify as modeling paradigm (e.g. General System Theory, Dynamical Systems, object-orientation). These modeling paradigms are related in various ways. In this thesis, GST with the notions of "system", "system boundary", "environment" etc. is regarded to be the most abstract modeling paradigm and associated with the specification of related notions. The notion of "Dynamical System" provides a mathematical concretization of GST, which is further concretized according to the general time-state relationship (see Figure 2.15). More concrete conceptual frameworks - referred to as "modeling paradigms" in this thesis - provide means for the specification of Dynamical Systems as "generative mechanisms" for experimentation, with different means to conceptualize causal structures. Thus, there is an apparent abstraction-specialization relationship between modeling paradigms, where relatively abstract paradigms provide less constraints (or no means) to the conceptualization of the different aspects of systems compared to more specific paradigms. The actual set of systems associated with a specific paradigm is a subset of the systems associated with the more abstract paradigm. This type of abstraction is further referred to as *paradigm generalization* (or *paradigm specialization*) and associated with the transition from the mere descriptive modeling of systems to the specification of explanatory models with causal structures.

Further modeling paradigms are related by means of *computation abstraction*, where the mathematical and computational meaning of a relatively "high-level" abstract paradigm is given in terms of a "low-level" paradigm (i.e. the computational semantics of the Stocks-and-Flows paradigm is given in terms of Differential or Difference Equations)[40]. Computation abstraction typically comes along paradigm specialization in that the relatively "high-level" paradigm restricts possible models to a subset of models that can be represented by the "low-level" paradigm by enforcing specific constrained thought patterns. These restrictions however should encourage the explicit specification of important aspects under consideration (e.g. feedback, locality of interaction, time-flow, spatial interaction etc.).

When a modeling paradigm is used to approximate another paradigm (e.g. Difference Equations approximates Differential Equations), the computational meaning of a paradigm is also given in terms of another paradigm. However, the approximated paradigm is not a computation abstraction in the sense that it restricts a more general conceptual framework to a specific set of high-level concepts. The paradigm used for approximating another paradigm rather provides concepts that are used to computationally mimic the semantics of models of the approximated paradigm, but does not give meaning to it in the

[40] Booch (2004) refers to this kind of abstraction as *virtual machine abstraction* where programming or modeling language abstracts from abstractions that built up the virtual machine.

sense of defining its semantics. Further, different paradigms might be related by means of *paradigm combination*, where a single paradigm is used to describe a specific structural or behavioral characteristic of a model that in the whole is described by means of several combined paradigms (e.g. combined discrete-event modeling, where ODE might describe continuous behavior and events/activities the discrete behavior, GAS where GIS concepts are used to describe structure and automata are used to describe behavior).

Although there are universal paradigms for the conceptualization of different basic classes of Dynamical Systems (e.g. DAE for continuous systems, Hybrid Automata and combined Discrete-Event for combined Dynamical Systems), the development of high-level paradigms shows that this universality often not the major motivation for the development of paradigms. Where universal paradigms set the basis for designing powerful simulation algorithms, which are based on universal simulation concepts, the use of high-level paradigms appears to be rather motivated by considerations of representation and pragmatics. Further, paradigms differ with respect to concreteness and ambiguity and many concrete characteristics are defined implicitly by tools and implementations of models (see 3.3.3).

The following Chapter 2.4 elaborates on procedural and epistemic aspects of simulation studies in order to set the context for further clarification of the roles and characteristics of modeling paradigms, associated modeling languages and tools in Chapter 3.

.

2.4 Methodological Background of System-theoretic Simulation Studies

This chapter provides an overview of main methodological and epistemological aspects of typical system-theoretic M&S studies by setting common experimental practices (Chapters 2.4.1, 2.4.2 and 2.4.3) in the context of the theory of model-based scientific reasoning (Chapter 2.4.4). The theory of model-based scientific reasoning provides a characterization of cognitive-epistemic background of M&S studies, that sets the fundamental framework for the design and evaluation of tools and languages, in particular of those aspects that typically restrain from formal evaluation (see Chapter 3).

2.4.1 Experimentation

In the practice of system-theoretic M&S, single issues or a set of seemingly related issues provide the objectives that are tried to be fulfilled within a single M&S study.

> *A M&S study is a set of tasks that are performed with the goal to meet objectives based on system(s), model(s), simulator(s) and experiment(s) given limited resources.*

Zeigler et al. (2000) differentiates three types of objectives of M&S studies:

- *Systems analysis* - the investigation of behavioral aspects of reality (e.g. for prediction)[41]

[41] In literature the term *systems analysis* is also used in a different meaning, when used to refer to a scientific discipline concerned with man-machine-environment systems (e.g. Chorafas (1965)).

- *Systems inference* - the discovery of unknown structural and causal features of reality
- *Systems design* - the investigation of alternative hypothetical future structures (e.g. designs of a structure).

Whereas, systems inference aims at gathering new knowledge in the sense of discovering unknown structures and causalities, systems analysis and design derive information from existing structural and causal knowledge about existing and planned systems. Irrespective of the type of objective of a study, a typical M&S study is characterized by the repeated execution of *experiments*.

> *An experiment is a controlled generation and collection of observations of a system and/or a model of a system. A M&S study encompasses the collection of observations and their successive evaluation for the purpose of characterization of the system or model.*

Experiments follow a common procedural template: according to the objectives of the study, the system is identified, a model of the system environment and the system with respective representations is produced, then the dynamical behavior in a given time span is derived by means of a simulation run[42] that is executed by the simulator[43]. This requires the specification of the initial state at the beginning of the studied time span. The model is finally evaluated, based on its structure and observed behavior. Based on the result of the evaluation the M&S process stops or repeats with a new experiment involving modified a model of the environment and/or the system. Several experiments might be grouped into *experiment series*.

> *An experiment series is a set experiments that are related by prescribed or observed structural or behavioral differences of respective models or models of the system environments. Evaluation is based on characteristics of experiments and experiment series.*

Experiment series might by defined by systematic variations of attributes of the models of the system and the environment, where variations might be defined algorithmically.

> *An automated experiment series is an experiment series that is executed automatically as prescribed by means of algorithmically defined variations of systems and models and system environments.*

An experiment allows for the characterization of systems only if sensible relationships between system, model and simulator can be established. Basically, this is framed into the assertion of the *validity of the model* and *correctness of the simulator*.

> Correctness *is the property of a simulator that it simulates the respective model with respect to all relevant aspects.*

[42] The term "simulation run" refers to the execution of the operations of a model by means of a simulator for a predefined modeled time span.

[43] Please note that in literature an "experiment" may refer to a set of related simulation runs. In this thesis, one experiment involves one simulation run, several related simulation runs are referred to as "experiment series".

Correctness is assessed by means of *verification*, which is part of the process of implementing simulators (Zeigler, 1984; Zeigler et al., 2000) . *Validity of the model* refers to the adequate agreement between model and system which is assessed by means of *validation* (Zeigler, 1984; Zeigler et al., 2000) . Zeigler et al. (2000) distinguishes three types of validity with increasing strength:

- *Replicative validity* states that the system's behavior observed given different inputs and initial states can generally be replicated by the model, but the association of one specific initial state and input with the exactly one specific observation is not possible [44].
- *Predictive validity* states that an agreement exists between model and system for a specific experiment with a specific initial state of system and model.
- *Structural validity* requires in addition to predictive validity that a model " [...] mimics the system in step-by-step, component-by-component fashion [...] (Zeigler et al., 2000)", thus the internal structure and behavior of the system must be reflected by the model to a degree that is imposed by the objectives of the study.

The evaluation of replicative and predictive validity is solely based on accessible observations, and thus can be treated formally by means of mathematical statistical data analysis to a great extent. In contrast, the evaluation of structural validity involves the evaluation of causalities based on the assumed internal structure of a system. This may go beyond the possibilities of direct observation, since the relevant structural features of the system may not be accessible for direct observation. Thus, whereas the assessment of replicative and predictive validity is widely based on mathematical statistical treatment of data (see Chapter 2.4.3), structural validity is additionally a cognitive and epistemic issue (see chapter 2.4.4). Verification is subject to formal analysis, but also subject to implementation practices (see Chapter 2.4.2).

2.4.2 Digital Simulation

A basic task of M&S studies is the implementation of simulators and their verification. In principle, the implementation of a simulator is based on the translation of the concepts of the model, that might be rather aligned with the concepts of the problem domain, into the concepts of target technology, thus digital computers that are characterized by discreteness, limited resources and possibly inherent serialism. Depending on the problem and technology used, there may be several related model formalizations with varying degree of orientation towards problem domain or target technology. According to (Overstreet and Nance, 1985), two types of model representation are commonly associated with the process of implementing simulators. First, *model specifications* that are rather used for human reasoning (i.e. analysis), documentation and communication. Second, a model representation that is used for setting up and executing the simulator - the *model implementation*. The model specification is derived from the mental model which should be based on a theoretically founded idea of structure and behavior. The model implementation is understood as being derived from the model specification, with possible intermediate representations.

[44] For any observed behavior of the system, there exists a model that replicates the observed behavior, but it is not possible to map a specific model to a specific output, e.g. if a model produced more than one observed behavior.

The simulator may exactly emulate the behavior of model. In case of fundamental differences between model and computing device (e.g. continuous state and behavior, parallelism), it may be required to approximate the model's behavior[45]. The quality of approximation is measured by means of the difference of expected correct behavior of the model specification and the simulator. If expected behavior and simulated behavior are recorded by means of respective mathematical structures, correctness can be assessed formally by means of formal similarity measures. The simulator is regarded to be correct, if the difference between model and simulator is not relevant in the context of the objectives of the M&S study.

However, where formal assessment is not possible, modelers approach verification by using "good programming practice" including software testing by re-implementing models with known behavior and compare this known behavior with simulated behavior. Further, modelers use visualization to verify that the simulator generates the expected behavior (Kleijnen, 1995). For a number of classes of models (paradigms) some aspects of correctness of simulators has been established theoretically (e.g. synchronization of different types of discrete-event models and numerical integration algorithms (see Zeigler et al. (2000))). In this case, an a-priory estimation of the quality of approximation is possible and respective M&S technologies internalize respective translations (e.g. numerical integration algorithms, parallelization/serialization routines). Thus in practice, verification is based on available a-priori knowledge about simulation-related properties of models, software development skills, visualization, data analysis and available verified implementations.

2.4.3 Digital Data Analysis

The practice of experimentation is in great parts concerned with collection and processing of input, output and observed data within experiment series. Single tasks of data analysis are commonly motivated by verification (evaluate similarities of expected and generated data), validation (evaluate similarities of observed and generated data), but also by the aim to understand a models behavior, irrespective of its representative function. Data analysis in experiment series typically aim at (Kleijnen, 1997):

- Comparison of data that is typically generated by a set of similar models (sensitivity analysis and parameter reduction).
- Identification of the model from a set of models with smallest difference between generated and observed data (calibration) or some other predefined characteristic (optimization).
- Compact characterization of set of models whose members differ due to inherent stochastic variability.

Thus, the data analysis involved in experimentation in great parts aims at derivation compact characterizing representations and the derivation of similarity measures. There exist a variety of methods for the derivation of compact mathematical representations of data (e.g. basic statistical measures, such as average distribution, counts and functional representations of time series) and respective procedures for their computation which is

[45] If the need to approximate influences the properties of the model specification, the quality of the approximation is not only subject to verification, but also validation. Such conceptual approximation is often motivated by limited computational resources, uncertainty of observations or the aim for simplicity.

subject to a problem domain itself. The discussion of these measures is beyond the scope of this thesis, however there are three basic characteristics relevant in this context:

- The measures that describe the similarity of data are usually based on concepts of morphisms between different data, thus data must conform to a well defined mathematical structure.
- The analysis of generated data depends on the domain and the objectives of a study. It ranges from the calculation of computationally relatively simple statistical indicators to the derivation of measures that might involve several interrelated distinct processing steps.
- Data analysis is usually performed by means of computers and often, at least partly, subject to automation by an algorithm.

In the presence of non-determinism, optimization, calibration and sensitivity analysis, the great number experiments typically requires the automation of experiment series within M&S studies. Thus the realization of experiment series involves the specification of all aspects of experiment series in a way that it can be executed automatically by the computer. Besides the algorithmic specification of data analysis, this involves the specification algorithms that constitute automated experiment series (e.g. stochastic variations, the traversal of parameter space and optimization algorithms). In practice, stochastic variations involve the use of pseudo random number generation algorithms when setting the value of a models parameters or variables initially or during the course of simulation. There exist a variety of random number generators, that deterministically generate random numbers according to given parameterization. For reason of minimizing the number of experiments in automated experiment series, the variation of a experiment may be dependent on the output a former experiment, so that the characteristics of the next experiment are calculated based on the output of former experiments.

Further, the realization of experiments and experiment series requires simulation control, including data retrieval, processing, visualization, persistence, and the initialization, execution and termination of simulators based on model implementations(Klahr, 1994; Zeigler et al., 2000).

Zeigler et al. (2000) proposes to formalize the conditions under which models and systems are observed or experimented with by means of a model referred to as the *experimental frame*. The experimental frame encompasses the specification of used input data, constraints and analysis of output (observations) and ensures that expected behavior (e.g. observations) can be compared to the generated behavior in a feasible way. In practice, automated experiment series also encompass the specification of data analysis, systematic model variation and optimization procedures.

2.4.4 Scientific Knowledge, Models and Type Hierarchies

Systems analysis and systems design build upon the fact that models represent knowledge about real systems. Systems inference aims at the discovery of knowledge about real systems. Thus, a key feature of scientific M&S studies is the relation of models and scientific knowledge, which is particularly subject to discussion when direct observation, thus formal similarity measurement, does not provide the means to constitute sufficient validity (uncertainty). In this case, additionally the constitution of relations between relevant physical or mental activities of M&S and existing knowledge is necessary to ensure

credibility and, from a philosophical perspective, high standards on what is to be regarded as "known" (Dunbar, 1999; Klahr, 1994; Aronson et al., 1994; Shapere, 1984).

The following chapters highlight the basic theoretical background of knowledge and models as used in M&S. In general, the epistemic characteristics of knowledge[46] are subject to current philosophical discussion. However, this chapter first presents basic features of knowledge, which can be accepted as a common denominator amongst approaches if "truth" is perceived as "believed to be true by sensible evaluation of all available information", but not necessarily as "truth" in the sense of "really such under all circumstances". This is followed by a short characterization of widely adopted "logicist view" in the sciences against the background of which the more recent "model-based view" is characterized as the base for relating procedures of M&S, respective modeling languages and tools.

Basic Characteristics of Scientific Knowledge

Basically, some state of affairs are designated as *reality*. The truth about reality is not known, but reality is the source of *observations* that scientists perceive and interpret as properties of reality[47]. Observations, however, only provide a partial picture of reality, but knowledge helps to interpret these observations in a useful way so that knowledge is supposed to give meaning to observations beyond the directly observable. Against this background, Schlick (2009) provides a general definition of knowledge:

> Knowledge *is the recognition of something known (e.g. a physical law), or assumed to be known (e.g. a hypothesis), in something new (e.g. an perceived phenomenon), where the known serves as an explanation for the unknown*[48].

Relating knowns and unknowns by means of scientific reasoning is basically concerned with valid abstraction and credible specification and by this, knowledge appears to be organized at different levels of abstraction, where higher levels hold relatively general knowledge (e.g. relevant for a relatively great number of processes), compared to the lower levels. *Abstraction* refers to the identification of relatively general concepts and relations that serve as explanation for relatively great number of observations, whereas *specification* refers to the application of relatively general concepts and relations to special cases by means of concretization and refinement. Concepts at the different levels are produced and related by means of scientific reasoning, which appears as a number of accepted operations on accepted representations of knowledge, which is embedded into accepted scientific procedures.

The characteristics of truth, the representation of knowledge and the characteristics of credible operations of scientitic reasoning are subject to philosophical discussion, however three commonly accepted basic requirements for scientific reasoning can be identified (Schlick, 2009; Popper, 2002; Aronson et al., 1994):

1. *Precision* - knowledge discovery presupposes the use of precise concepts, since only precise concepts can be recognized. *Concepts* are precisely defined by means of their characteristic features, so that they can be referred to in an exact and unambiguous

[46] Scientific knowledge is further referred to as "knowledge".

[47] The terms 'system', 'phenomenon', 'process' or 'characteristic' usually refer to distinct parts of reality of interest as the subject of studies.

[48] An example for knowledge discovery is the recognition of the relationships exhibited by of phenomena of light to be the same as those that occur generally in the propagation of waves (Schlick, 2009).

way. As such, concepts represent all those entities (real world objects or concepts) that possess the features attributed to it. Knowledge appears as representation of concepts and the relations between concepts. Mathematical representation is commonly regarded as a way to precisely formalize concepts and relations (Schlick, 2009).

2. *Empirical adequacy* - to be regarded as knowledge, these imagined complexes of concepts and relations must provide means to explain observations and then, can be attributed to instances of unobserved phenomena (e.g. instances in the future). In the empirical sciences, modeling and experimentation serve at least the purpose of evaluating the empirical adequacy of assumed knowledge (Popper, 2002).

3. *Objectivity* - knowledge must be objective, thus it must not depend on personal presuppositions of single persons (Popper, 2002). Objectivity is realized by the inter-subjective evaluation of ideas within the scientific community, thus it based on scientific discourse and its characteristics (Schlick, 2009; Harré, 2004; Popper, 2002).

Although there is widespread agreement on these general characteristics of knowledge, the concrete characterization of knowledge representation and the role and characteristics of experiments and models in particular is subject to philosophical discussion, in particular with respect to creative processes of knowledge discovery[49]. Two main opposing viewpoints can be identified in this context: 'Logicist' and 'model-based' approaches to philosophy of science.

Logicist View

The "logicist approaches", which amongst others subsume the influential works of the "Vienna circle" (e.g. Schlick (2009)) and Karl Popper (e.g. Popper (2002)), presume a mathematical axiomatic representation of knowledge within a theory. Scientific reasoning is based on formal inductive (abstraction) and deductive (specification) reasoning in order to relate concepts that are embodied in the axioms of a theory (e.g. Popper (2002)). Concepts must be formalized in the context of existing theory, which is represented by axioms, as a hypothesis which is further tested by means of the evaluation of its empirical adequacy. For this, formal deduction is applied to produce models that serve for experimentation that serves for testing the empirical adequacy of a hypothesis, thus a new theory. In this approach, models are structures that make all sentences of an axiomatic theory become true. Thus models are perceived as an interpretation of an abstract calculus (Frigg and Hartmann, 2009). In this view, models do not represent, but rather specify knowledge at lower levels of abstraction. The rigorous formal character of scientific reasoning ensures that models and theory are consistent in a verifiable way, thus that models are tightly related to the axioms of a theory, thus existing knowledge.

However, logicist approaches do intentionally not reflect the history and creative part of theory building (Schlick, 2009; Popper, 2002), thus an important part of scientific practice that is the context M&S and the technology used herein. The logicist view is particularly criticized to neglect the importance of models (Harré, 2004; Shapere, 1984; Hjørland, 2004; Aronson et al., 1994) . Attempts to include historic aspects and cognitive processes of scientific knowledge discovery in theory, have lead to the view that models precede

[49]The open philosophical question if knowledge refers to a reality that really exists ('truth'), or to a reality that is just strongly believed to exist, is ignored in this text.

theory development in the logicist sense and that knowledge representation and discovery is primarily based on models (Frigg, 2006; Nersessian, 1999)[50].

Model-based View

According to the "model-based view", models are the main means to discover and represent knowledge. A basic characteristic of this view is that knowledge is represented by means of models that are instantiations of types which are organized within a type hierarchy. There are two basic views on such type hierarchies that both highlight different kinds of relations between types and between types and models: the formal view and cognitive view. Figure 2.26 illustrates the idea of type hierarchies at the example of harmonic oscillators with respect to oscillation period (τ, from Aronson et al. (1994)).

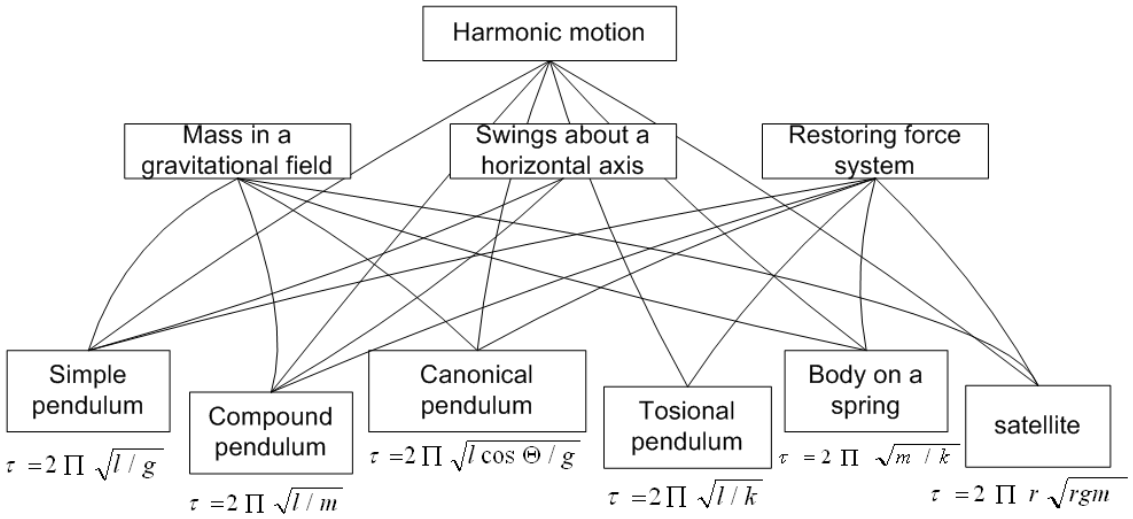

Figure 2.26: An example type hierarchy for harmonic oscillators (from Aronson et al. (1994)).

Formally, a type is an entity with a number of properties and invariant relationships among these properties. A type prescribes properties that are numerically identical among its subtypes ("first-order properties") and properties that may vary among the subtypes ("second-order properties"), thus types describe quantitative and qualitative similarities of subtypes. Each type specifies further constraints on the relation of its properties that are valid for all subtypes by means of "meta-properties". On the one hand, meta-properties are empirical abstractions of observations that may be thought of as a propositional law (Aronson et al., 1994) and on the other hand, they " [...] function more as recipe for constructing models than like general statements (Giere, 1994)."

In the case of an entity having only one supertype, the subtype specifies boundary conditions for the supertype by concretization of second-order properties and/or addition of further properties. In the case of multiple supertypes, the laws of all supertypes have to be combined to form the law of the subtype. Types are related by a an "inheritance" relationship, which states that each property of a supertype is a property of all its subtypes.

[50] In this model-based approach to scientific knowledge discovery, models may, but do not have to be derived deductively from theory as in logic-based approaches (Frigg, 2006).

In a type hierarchy, models are instances of types, thus materializations of types with all properties of the supertypes specified concretely so that it allows for experimentation[51]. Such model might represent a specific system (= local model) or a number of systems to which the properties of the model are attributed to (= class of models)[52]. Although type hierarchies resemble semantic nets they are to be understood to be more general since they are the framework for scientific model-based reasoning by scientists that is understood to be partly, but not exclusively, formal (e.g. expressable by an inference algorithm, (Aronson et al., 1994; Harré, 2004), see below).

The cognitive view of on type hierarchies refers to the characteristic that well known basic abstraction and specification methods relate models and types, where the characterization of new elements usually is the result of a combination of a number of these basic techniques. Common abstraction techniques encompass (Nersessian, 2002b; Nersessian and Patton, 2009) :

- *generalization* - inductive attribution of common characteristics of a set of subtypes and models to a supertype,
- *limiting case abstraction* - stepwise reduction of influence of a factor,
- *idealization* and *generic abstraction* - suppression of specific factors in order to find isomorphisms and achieve generality,
- *aggregation* of characteristics of a set of types and models based on an evaluation of differences (e.g. averaging, exemplification) and
- *functional abstraction* - definition of a supertype by characterization of its function (e.g. the type "pump" as supertype of "heart" and "water pump").

In contrast, specification is associated with the application of the knowledge that is contained within the type hierarchy. Common operations are to be applied such that the properties of the supertype or model are not violated:

- *refinement / correction* - substituting and complementing general properties and relationships by means additional or more specific, properties and relationships that explain or amend general properties,
- *concretization* - limiting possible occurrences and behavior and
- *simulation* - generation of observations.

Indeed, there is a variety of formal methods and experimental techniques that support the evaluation of type membership (e.g. statistical aggregation) or production of members (e.g. experimentation procedures) by exploiting formal aspects. However, it is a characterizing feature of model-based scientific reasoning that it cannot be reduced to formal aspects. Informal scientific reasoning is particularly relevant when creative problem solving is applied in the case of knowledge discovery with uncertainty (Nersessian, 1999). In particular formal (experimental) methods do not explain the location of the boundary

[51] Aronson et al. (1994) and Harré (2004) use the term "model" in a sense that corresponds to the term "model representation", in particular the simulator, in this thesis.

[52] Please note that a model is often used to represent theory at a higher level, if it describe properties of a number of models (Frigg and Hartmann, 2009; Hughes, 1997).

of classes that are prescribed by types or the levels of abstraction used productively in practice. Cognitive aspects of informal model-based scientific reasoning appear to account for this (Nersessian, 1999; Giere, 1994).

Further, type hierarchies and the properties of types are specific to problems, thus different type hierarchies might be adequate to represent the same state of affairs, depending on the purpose of reasoning. In practice, type hierarchies are usually not directly represented. Types might denote and intermingle features that are believed to exists irrespective of human intervention (objective features) and entities that depend on convention (conventional features, Aronson et al. (1994)). Types that exist across different type hierarchies and which are used in different contexts are believed to denote "natural kinds", thus structures and laws that are believed to truthfully exist.

Model-based Reasoning and Type Hierarchies

In M&S, reasoning processes are embedded into experimentation procedures. This section provides a characterization of basic cognitive aspects that are attributed to model-based scientific reasoning. From the cognitive perspective, it is the basic characteristic of models and types, that they represent imaginable (hypothetical) mechanisms that provide sensible means for manipulation, simulation and communication. Following Nersessian (1999), this type of scientific reasoning is further referred to as *model-based reasoning*.

Giere (1994) elaborates on the cognitive aspect of type hierarchies by reference to observed descriptive properties of abstractions, which are laid out in more detail for the general case in Rosch (1978):

- The *graded structure of types*: the models that constitute a type are grouped because of their cognitive similarity. Cognitive similarity appears to be measured relative to an exemplary prototypical 'focal model' that exemplifies the basic characteristics of a type and constitutes the greatest difference to contrasting types. Operationally, cognitive similarity appears by means of common attributes, by means of common motor interaction, similarity of (imagined) shapes and the identifiability of averaged (imagined) shapes (Rosch, 1978). Focal models, in contrast to peripheral models, are relatively simple (highly idealized) and have a relatively great number of applications. There is typically great degree of agreement on the characteristics of focal models in contrast to the boundaries of a type (Rosch, 1978).

- The *vertical structure of type hierarchies*: In each type hierarchy, there is a level that appears to be most amenable to reasoning: the "basic level". Different types at the "basic level" appear to be easily distinguished and their concepts relatively easily learned. Further basic levels models are the first to be applied to systems and understood. Members of a type at the basic level are (cognitively) more similar than models at higher levels, while models of lower level types are not significantly more similar. For basic level types it appears that it is possible to form mental images that are reasonably representative (isomorphic) to all members of a class. A basic level is not characterized by internal characteristics of models, but "various cognitive interactions between human agents and the real systems these models represent (Giere, 1994)."

These characteristics do describe, but do not explain properties of type hierarchies. Rosch (1978) counts responsible two "principles of categorization": cognitive economy

and perceived world structure. "Cognitive economy" means that a classification system provides a maximum of information with the least cognitive effort given perception and the purpose of the category system. "Perceived world structure" refers to the assumption that perception of human agents with knowledge is structured in a way that perceived structure is such that some known combinations of characteristics are more likely to occur than others, thus the perceived structure is not arbitrary but aligned with existing perceptions (Rosch, 1978). These properties of classification can be set into the context of M&S against the background of specific cognitive processes in scientific knowledge discovery using models and experimentation that refines this characterization of type hierarchies.

In general, each model is related to a subject (target), the modeled phenomenon represented by observations, and a source that provides explanatory characteristics of the model that are believed to exist (Aronson et al., 1994). Source and target of a model might be identical so that straightforward abstraction techniques relate source and model, thus system and model (Aronson et al., 1994). In case of high uncertainty, creativity, thus more complex reasoning is applied, where the source of a model might be associated with other domains or the type hierarchy itself. Although in case of uncertainty scientific reasoning appears by means of basic operations presented above, the following characteristics underly the reasoning process, particularly in the context of experimentation:

- *Analogical reasoning*: in analogical reasoning the source and the target of a model differ. In general, modelers combine characteristics from a source and the target (domain) in order to derive new types and models, where the source (e.g. a hypothetical mechanism) is thought of existing. A prerequisite is that source mechanisms must be represented at a level of generality that allows for the deductive generation of class members (Nersessian, 2002a). The choice of characteristics of source and target and the method of combination is based on domain knowledge and not based on formal reasoning . Generic modeling, thus the modification of demonstrative models, thus specific models that represent most important features demonstratively is a basic cognitive technique. Visual reasoning, the modification and simulation of schematic internal imaginistic iconic mental models is fundamental, since it enables to bypass constraints of other (linguistic, formulaic) representations, while conforming to, possibly implicit, domain specific constraints (Nersessian, 2002a).

- *Exploratory approach*: modelers iteratively define different causal mechanisms or variants of mechanisms by means of mental models and evaluate them by means of experimentation. This model-based exploratory approach to scientific knowledge discovery is based on cognitive mechanisms of perceptual-motor activity, thus based on interaction (interact - perceive - modify) between model and human, rather than formal reasoning. Exploration however is tightly constrained by available scientific knowledge (Nersessian, 2002a).

- *Interlocking models*: models and their representations are the means to interlock different aspects of scientific reasoning: theory, experimentation practices, perception, technology and discourse. Besides theory, the use of technology for experimentation introduces a variety of additional constraints and the used technology influences thought patterns, that are embodied in models. As the main artifacts of scientific discourse, models also carry properties that reflect the need of communication. Thus, models and their representations are a vehicle to integrate the main theoretical and practical aspects of scientific reasoning processes (Nersessian and Patton, 2009).

- *Distributed reasoning*: models are subject of discussion within a group of scientists, particularly in decisive and productive moments of studies and for evaluation of objectivity. The prerequisite for successful scientific discourse that all participants have the same mental model about the mechanism under discussion, which is to be derived from perception and communication (Dunbar, 1999).

Aronson et al. (1994) illustrates the characteristics of sound scientific reasoning and the possibility of formal systems to document, but not to substitute human reason, at the example of modeling atoms at the analog of solar system: what makes the solar system a fruitful analogue of the atom is that knowledge about the internal workings of these systems has been established that is represented by the common supertype *Central force field* (Figure 2.27).

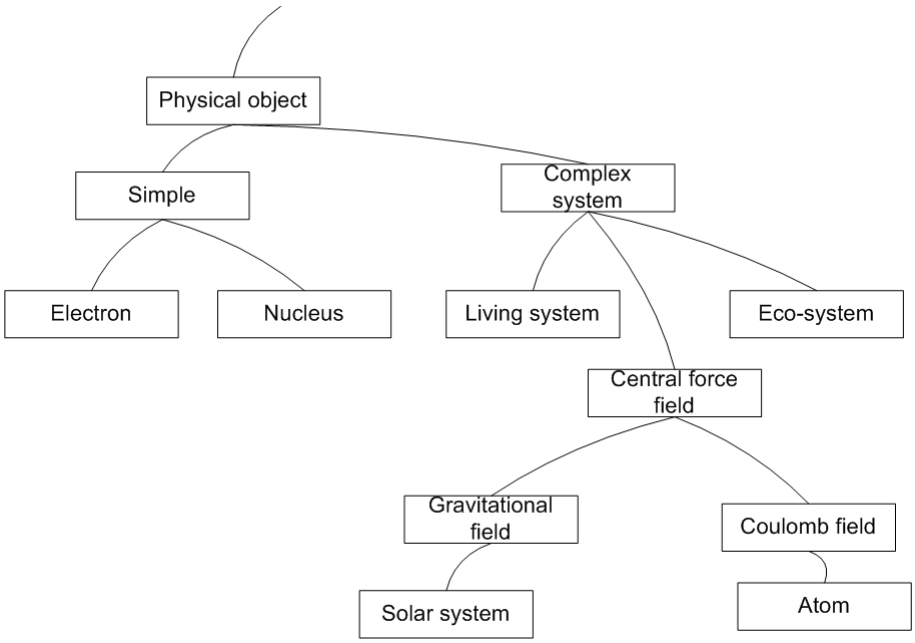

Figure 2.27: A partial type hierarchy (from Aronson et al. (1994))

In contrast, a disfunctional analogue (e.g. racing car system, Figure 2.28), although it matches formal similarity measures, would not allow the definition of a common supertype since its internal workings are not similar Aronson et al. (1994).

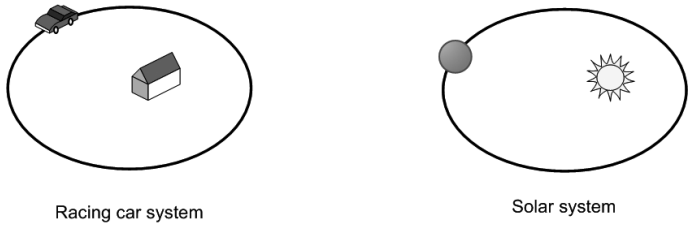

Figure 2.28: Racing car on elliptical track as an inadequate analog for the solar system (from Aronson et al. (1994)).

In this account, type hierarchies reflect logical mathematical reasoning and cognitive

mechanisms of reasoning, with models as the tangible mediators between theory and practice. In M&S models are accessed via their representations. The characteristics of representations in reasoning processes is characterized in the following.

Representation of Models

External model representations, thus model specifications and simulators, are central artifacts of reasoning processes and their characteristics influence the degree to which reasoning processes are supported. From a cognitive point of view, model-based reasoning can be seen as a process of interaction of humans agents with models via their representations. Interaction consists of synthesizing chunks of knowledge into internal representations - mental models - , the transformation of these models into external representations (e.g. for analysis, simulation and communication), interpretation of results and synthesizing the results into the mental model. The main reasons for using external representations are (Kirsh, 2010):

- *Cost efficiency*: Different representations allow the interaction on models with different costs (e.g. modification, simulation, analysis). Besides functional possibilities (e.g. enabling rearrangement of complex structures), it is the constraints (e.g. physical contraints) that affect cognitive costs of usage.
- *Synchronization of mental models*: External representations are the base of sharing mental models.
- *Persistence of models*: "Both rearrangement and having stable objects to think with both rely on physical things being persistent. [...] The brute fact of physical persistence, then, changes the reliability, the shareability, and the temporal dynamics of thinking (Kirsh, 2010)."

The quality of external representations depends on the functionality (e.g. simulation, analysis) it supports and the efficiency with which the necessary interaction is supported. Whereas the supported functionality (manipulation, simulation, communication etc.) is a characteristic of a representational format (e.g. text, equations, graphics, material) as it exists, the efficiency is a characteristic of the transformation of informational content that is used for reasoning and communication between internal and external representation[53].

Transformation costs are relevant for scientific reasoning because they may cause incorrect extraction of informational content, thus the construction of correct mental models depends on them and as a consequence correct individual and distributed reasoning. Further, the speed of transformation limits the ability for representations to serve as vehicles for thought in learning processes: "If external manipulability matches the internal requirements on speed, then the external medium has the plasticity to be a candidate for thinking in (Kirsh, 2010)."

The characterization of representational formats with respect to efficiency of transformation can be made against in terms of explicitness of informational content. Although there is no precise and commonly accepted definition of "explicitness of representation", (Kirsh, 1990) argues that common intuitions can be captured when explicitness is considered on the grounds of the computational cost caused by the extraction of informational

[53] In the context of M&S studies, depending on the task to execute, the informational content might anything from the state trajectory of a dynamical system to the underlying structural and behavioral characteristics of the model.

content from a representation to a form that is "readily usable" by the interpreter. Informational content is information needed by the interpreter for the task at hand. Thus, informational content is specific to the knowledge of the interpreter, the subject under consideration and the cognitive tasks (e.g. rearrangement, simulation etc.) to execute.

Cognitive computational costs of transformation range along a continuum from explicit (all informational content is immediately available at minimal computational costs) to completely implicit (informational content is not recoverable without additional knowledge; Kirsh (2005)). Following Kirsh (1990) costs fall generally into two categories. First, there are costs caused by the need to access identify relevant representational elements (process view) in a representation (e.g. find a datum in text). Second, there are costs caused by those computations needed for extraction of relevant information from representational elements (e.g. compute "2 + 2" instead of "4", structural view). More concretely, general conditions for explicit representation might be defined as follows (Kirsh, 1990):

- Representational elements (e.g. expressions) that explicitly encode information must be easily separable (e.g. not spatially distributed/separated in text or graphics).
- In ambiguous languages, where the meaning of an representational element must be derived from context (context sensitive), it must be trivial (= take constant time) to identify the syntactic and semantic identity of a representational element.
- Representational elements that explicitly encode information must be either readable in constant time completely or at least those parts of a representation must by immediately recognizable that serve the purpose at hand (e.g. evenness of a number (arabic) can be evaluated by last digit only). Immediate recognizability means that a representational element fits into the attention span of the interpreter and that it can be matched in memory directly.
- The information encoded by a representational element is immediately given, thus it directly activates respective usable internal mental representation.

From the above characterization of representational format follows that: Cognitive costs rise with representation of information not relevant for the cognitive task at hand, since they have to be processed. Cognitive costs rise with perceptual decomposition and separation of the representation of informational entities and indirections. Cognitive costs may fall when representations are tailored towards the operations performed on derived informational content. Cognitive costs fall with the degree the representation immediately activates internal representations ("structures, states, processes") used for reasoning.

2.5 Conclusions

M&S studies involve numerous basic technical challenges. Tools for M&S are primarily perceived to serve the purpose to deal with these technical challenges. Ultimately however, tools serve the goal to support epistemic and cognitive processes such that both creativity and credibility are combined with mastering technical challenges. Investigations related to model-based science clearly show that modeling modeling and the representation of models and corresponding tools are part of a cognitive system. Related reasoning processes have been presented at the base of the notion of type hierarchies, that are formed through

reasoning operations that are only partly formal. It is argued in this thesis that these kind of type hierarchies provide a general background against which modeling tools and language can be designed and evaluated.

2.5.1 The Role of Modeling Tools

As "enablers" of M&S studies, tools are required to enable the implementation of correct simulators as a prerequisite for repeatable simulation experiments and experiment series. This indeed refers to general aspects of computation and approximation, but also to the support of the development of simulators as software, to which issues of software development and characteristics of programming languages and corresponding tools generally apply (pragmatics, see Chapter 3). Basic necessary functionality provided by tools is the execution of simulation runs complemented by processing of input and output data, possibly within automated experiment series, which includes the automated setup and execution of different simulators. A basic non-functional requirement is efficiency in the sense that tools enable M&S studies within the limited given resources time, hardware and technical skill.

Against the background of the model-based view, it can be argued that the characteristics of modeling tools influence the quality of scientific M&S studies substantially beyond purely enabling technical considerations. The presence of uncertainty in particular requires creative problem solving under the premise of ensuring scientific soundness. Creativity appears to restrain from explanation through formal reasoning, thus scientific reasoning with purely formal relation of theory and models. The model-based view suggests to refrain from sole reliance on formal reasoning for the sake of creativity and instead to cast the assessment of credibility in terms of type hierarchies and associated reasoning processes. This requires to take cognitive aspects of experimentation into account.

Empirical investigation of the use of modeling technology in laboratory environments (Harmon and Nersessian, 2008) suggest that "technology plays a greater role than simply as a tool for offloading some task." Harmon and Nersessian (2008) concludes that "the central purpose of the technology is often not what it 'does' for the researcher, but how it enables the researchers to embody and test hypothetical models [...]. We use the notion 'cognitive partnering' to capture our observations that the researchers understand and interact with the technology they design and construct as though they were collaborators in research."

Characteristics of tools that support creative scientific reasoning may be identified from the context of model-based reasoning.

2.5.2 Types and the Representation of Models

The characteristics of model representations influence the cognitive costs of switching between different representations. The lower these costs, the more likely M&S tools support "short" motor-activity cycles that are based on the immediate reception of feedback to action (e.g. the modification of a generative mechanism). Such motor-activity cycles constitute a basic element of exploratory model-based scientific learning. It appears that exemplary demonstrative imaginable mechanisms that can be visualized and manipulated in various ways simultaneously (mentally, visually, formally etc.) are particularly productive in model-based reasoning. However, imagined mechanisms must fit into the framework of knowledge that is perceived to be established by means of type hierarchies and respec-

tive types must provide adequate degrees of freedom for the generation of subtypes by means of modification. The compromise between concreteness and freedom depends on the subject and problems under consideration and the existing knowledge. From a cognitive perspective, it appears to be relatively difficult to identify the degrees of freedom, compared to the identification of the core imagined mechanisms.

For productive model-based reasoning with several simultaneously accessed representations, representations should be directed towards cognitive images of the hypothetical mechanisms under investigation, taking into account that different researchers must derive the same mental models from representations, which is fundamental for both, creativity through distributed reasoning and scientific soundness through interpersonal evaluation (validation), in particular when the relations drawn to knowledge are not formalized to a widely accepted degree. However, the basic requirement of (automatic) derivation of simulators and data processing programs requires to take considerations of simulation and data processing technology into account when conceptualizing systems and experiments. Thus at this point representations of models interlock epistemic, cognitive, technical and procedural aspects of M&S.

2.5.3 Common Levels of Abstraction in M&S

Although levels of abstraction are generally arbitrary and sensible levels of abstraction are not predictable in general, some levels of abstraction appear to be well-established in the field of system-theoretic M&S. In general, any conceptual framework that provides "forestructures" for reasoning about reality might be interpreted as a type as long as its properties can be cast into first-order, second-order and meta-properties of types and if valid relations to other types can be drawn based on accepted operations of scientific reasoning (see Chapter 2.4.4). As a relatively abstract conceptual framework GST (Chapter 2.2.1), prescribes basic structural and epistemic characteristics of systems that find their mathematical concretization in Dynamical Dystems (chapter 2.2.2), as opposed to non-mathematical system modeling. The different classes of Dynamical Systems (continuous, discrete, combined, chapter 2.3.1) define different subsets of dynamical systems by means of relatively general assumptions about states and their relation, that induce different relatively concrete mindsets for the observation and conceptualization of systems (e.g. discrete vs. continuous time).

By incorporation of specific assumptions about the structure of systems, modeling paradigms (Chapter 2.3.2) provide conceptual frameworks for the conceptualization of systems as generative mechanisms that allow to directly relate assumed or known structures and causal relationships with observations. Relating modeling paradigms by means of paradigm specialization and computation abstraction (Chapter 2.3.3) is a common technique to introduce and enforce assumptions about the structure of modeled systems and to give exact mathematical and computational meaning to high-level abstractions. Relatively universal paradigms have been subject to theoretical considerations, in particular with respect to analysis and simulation, so that universal simulation technologies can be built upon a sound theoretical basis (see Chapters 3.3 and 3.4). The major motivation for the usage of more more specific paradigms is that they allow for other kinds of formal analysis (e.g. constraint checking) and that these paradigms provide templates for the construction of mechanisms according to a specific common conceptual framework that fits the requirements of model-based scientific reasoning relatively well, thus particularly supports cognitive aspects of modeling.

A further distinct level of abstraction, that is naturally introduced by the concept of Dynamical Systems, is formed by parameterizable models, thus model specifications, where the values of parameters are not specified concretely. Parameters typically represent aggregate properties that are not modeled at the intended the level of abstraction. Specification, thus setting of parameters might be made at the base of knowledge or at the absence of knowledge, the values of parameters might be set experimentally (e.g. optimization). Observations explicitly represent specific systems within a particular time span as observed at the system boundary.

Tools for M&S provide concrete means to specify models by means of computer languages. Typically, model specifications and representations of observed data are the type of model representation the modelers directly interact with. The following chapter provides a characterization of theoretical and practical background of computer languages and tools for M&S and EMS and further identifies the characteristics of MDE that are finally set in the context of model-based reasoning for the design and evaluation of the application of MDE in EMS.

3 Computer Languages and Tools for M&S and EMS

Computer languages are historically primarily associated with the development of software and software-intensive systems and software engineering. The clarification of related terms, current issues, solutions in this Chapter forms the background for the design and evaluation of the application of MDE to EMS. There is a substantial theoretical background related to the design and specification of computer languages and the implementation of respective tools (Chapter 3.1 and 3.2). The presentation of existing tools and computer languages for M&S in engineering and EMS (Chapters 3.3 and 3.4) concretizes general aspects of M&S and EMS, in particular with respect to technical issues and solutions. Against this background, Chapter 3.5 presents Model-driven Engineering. Chapter 3.6 concludes Chapters 2 and 3 and identifies the possible role of MDE in the context of EMS.

> *Informally, a computer language is a language that provides humans the means to encode informational content that is to be processed by digital computers.*

The use of computer languages is based on *language tools*.

> *A language tool is software that facilitates humans the production of valid specifications according to a computer language and to realize the informational content of specifications by means of computation (e.g. to execute a calculation).*

In general, computer languages are designed to solve problems with given limited resources. Thus in general computer languages are designed and evaluated at the base of the functionality in terms of possible computations and the costs associated with the implementation of language tools, implementation (including maintenance) of programs using language tools and the execution of programs. Technically, language tools typically encompass at least these components: an *editor* provides users with the means to produce specifications, the *compiler* reads specifications and translates them into an executable representation, a *linker* links several executable representations in case there are dependencies between them. The execution of the specification is typically realized within the execution environment provided by the target computer's operating system or an interpreter. Although, a simple text editor, a plain compiler/linker and an operating system are theoretically sufficient for the usage of computer languages, the state-of-the-art tool support provides additional functionalities that are perceived to play a decisive role for the success of computer languages, such as syntax highlighting, code completion in editors and tools for debugging, profiling and testing[1].

[1] A collection of tools that makes up a language tool is typically referred to as *Programming Environment*. When different functionalities are tightly integrated in one tool, the tool is typically referred to as *Integrated Development Environment* (IDE). In this thesis, the term *language tool* refers to both programming environment and IDE.

The theory of computer languages as developed in the area of programming languages is in large parts based on considerations about efficient compiler technology, whose theoretical foundations are the base for many functionalities of language tools. A compiler typically consists of distinct parts (Parr, 2009):

- The *reader* reads the specification and recognizes, if the specification conforms to the syntactical rules of language. The reader usually builds an internal syntax-oriented representation of the specification - the *parse tree*.
- The *semantic analyzer* reads the parse tree and derives an abstract internal representation tailored towards further processing (e.g. checking static semantics, execution, code generation) - the *abstract syntax tree (AST)*.
- The *generator* generates an external syntax-oriented representation (e.g. machine code) that is based on the abstract internal representation (i.e. AST).

Computer languages are typically characterized with respect to three interrelated aspects:

- *Syntax* is concerned with the available concepts, notational elements (alphabet) and the rules of their combination (syntactic structure of sentences).
- *Semantics* is concerned with meaning of syntactically correct specifications in terms of non-syntactical constraints (static semantics[2]) and computation (dynamic semantics).
- *Pragmatics* is concerned with practical aspects of the usage of a computer language by humans.

Technically, the specification of syntax and semantics are sufficient for the implementation of computer languages by means of respective language tools and amenable to a relatively great degree of formal treatment and automation. In contrast, pragmatics is regarded to elude from formal treatment to a great degree and typically treated informally, separately for syntax and semantics. However, pragmatics have profound influence on the design of computer languages.

The following Chapters present syntax and semantics of programming languages and their specification and implementation, followed by a presentation of key aspects that are commonly related to pragmatics: abstraction, programming language paradigms and regularity.

3.1 Definition of Computer Languages

The implementation of language tools requires that the sentences that form a specification can be recognized algorithmically and that informational content of sentences is unambiguously represented by computations of the receiving computer. The assertion of these requirements and the efficient implementation of language tools is subject to syntax and semantics.

[2] In practice, static semantics and syntax are processed together, so that static semantics might be seen a part of syntax.

3.1.1 Syntax and Semantics

Formally, a computer language is commonly defined as the set of valid sentences (finite sequence of symbols) over an alphabet that contains the symbols of the language (Harrison, 1978; Parr, 2009). This definition covers syntactical considerations only and appears to be appropriate for considerations related to specification of syntax and the efficient recognition of valid sentences. However, computer languages can be more broadly defined as a structure consisting of syntax and semantics (Harel and Rumpe, 2004). Terms are clarified in the following.

Language theory distinguishes between *concrete syntax* and *abstract syntax*. Whereas concrete syntax refers to the actual form of representation of syntactical elements arranged in strings or graphical primitives as used by humans and recognized by the reader, abstract syntax refers to the internal representation of a specification in the computer that is tailored towards processing the specification by directly representing the compositional structure (e.g. by removing purely syntactical elements and representing composite syntactic elements directly as trees or graphs (Harel and Rumpe, 2000)). The concepts of the abstract syntax are typically closer to the model of the CPU and tailored towards asserting non-syntactical context-sensitive constraints (static semantics), optimizing and processing of dynamic semantics. The Abstract Syntax Tree (AST) is usually derived from concrete syntax (Syntax Tree), e.g. by removing purely syntactical elements. Further, attributes or additional data structures (e.g. symbol table, scope tree) are derived that add references between elements of AST for checking static semantics or realization of dynamic semantics, such that the AST is used as an Abstract Syntax Graph (Parr, 2009; Mosses, 2006). The use of formal grammars is state-of-the-art in defining the syntax of programming languages (Chapter 3.1.2).

Dynamic semantics is theoretically given by means of a mapping of the syntax elements (usually abstract syntax for non-trivial languages) onto a semantic domain that defines their meaning (e.g. the symbol "+" means addition of what is referred to left and right from it). In practice, several approaches exist to describe dynamic semantics of a computer language. Typically, a basic distinction is made between informal specification by means of natural language within documentation and formal methods, where "formal" means that the specification of dynamic semantics is written in a language that already has a precise meaning (Mosses, 2006). Unambiguous mathematical documentation, formal analysis of languages and automation of language tool implementation are primary goals of formal specification of dynamic semantics (Chapter 3.1.3). A computer language with formal specification of syntax and semantics may be referred to as *formalism*. This thesis is concerned only with computer languages that allow for specification of executable programs, thus syntax and semantics must be given in an unambiguous way, at least in form of working language tools. The term "computer language" and "formalism" are used interchangeably.

3.1.2 Formal Grammars

In practice, context-free grammars and respective formalisms (e.g. Backus-Naur Form (BNF)) appeared as particularly useful for implementing language tools for textual languages, since they allow for theoretically well-understood efficient recognition of specifications with correct concrete syntax, while covering a reasonable range of syntactical requirements and allowing relatively efficient implementation of tool support, e.g. through

automated implementation of lexers and parsers (Parr, 2009). In its simplest form, a formal grammar specifies a set of terminal symbols (literals), non-terminals (aggregate symbols) and a set production rules that define how non-terminals are composed from symbols[3].

However, some characteristics of context-free grammars, as defined by BNF, appear to be disadvantageous:

- Serial concrete syntax: grammar rules might not only reflect concepts of the languages, but also issues of serialization, such as precedence and dangling-else. Further, formal grammars are tailored towards textual languages, not graphical languages that are not serial in character.

- Concrete syntax orientation: the relationship between abstract syntax and concrete syntax is specific to language tools and their implementation (e.g. programming language, grammar specification language), thus the concepts of the language are not defined by the context-free grammar only (Parr, 2009)[4].

- Tree structure of syntax: non-trivial languages usually encompass context-sensitive constraints (static semantics), such as scopes, that cannot be expressed with context-free grammars. Thus, these constraints are implemented manually in the semantic analyzer of the language tool, often even dependent on features of the programming language used (Feilkas, 2006; Parr, 2009).

- Lack of refinement and abstraction mechanisms: context-free grammars do not allow the specification of language constructs as the refinement of other abstract language constructs (Krahn et al., 2008; Fischer et al., 2004).

3.1.3 Dynamic Semantics

Several methods exist for the formal specification of dynamic semantics serving different purposes (Mosses, 2006):

- *Operational semantics*: Abstract syntax is mapped onto operations of an (abstract machine). This approach appears natural for building language tools, since executing dynamic semantics requires translation into machine language. Tool support (e.g. interpreters) can be derived from specification. Several variants exist: (Modular) SOS, Reduction Semantics, Natural Operational Semantics, ASM.

- *Denotational semantics*: Abstract syntax is mapped onto mathematical objects, typically domains and functions over them (e.g. Lambda-notation). Denotational semantics are designed to provide proper mathematical foundation for reasoning about programs and understanding of programming languages. Tool support (e.g.

[3] The practice of language tools for textual languages distinguishes between characters (e.g. 'a', 'b','1') and tokens (e.g. 'while'), where basic characters are used to form tokens that are used to construct sentences. In language tools, the reader component itself is usually composed of two components, the *lexer* that identifies tokens in the stream of characters and the *parser* that identifies the syntactical structure of sentences made up of tokens.

[4] Although formalisms such as attribute grammars and graph grammars allow for tool independent specification of context-sensitive constraints and automatic derivation of tool support, they have not found comparable acceptance in practice. This may be due to difficulties in usage.

interpreter) can automatically derived from specification. Denotational semantics is mainly applied in teaching and research, but rarely used for the specification of larger programming languages.

- *Axiomatic semantics*: Abstract syntax is mapped onto axiomatic rules describing assertions about values of variables before and after execution of each construct. The main aim of axiomatic semantics is the verification of properties of programming languages, which has been applied in some cases only.

Although formal methods appeared generally to be applicable for the purposes they have been intended for, in particular the derivation of theoretical properties of computer languages and the clarification of language concepts, there are issues that apply to varying degrees to the different methods and their variants that appear to object their widespread use in the practice of the definition of computer languages and the development of language tools: the monolithic and complex character of descriptions, thus limited maintainability and extensibility (e.g. SOS, Lambda-notation); tediousness (e.g. Reduction Semantics), the complicatedness (e.g. Axiomatic) and limited comprehensibility (e.g. Denotational semantics) of specifications of practical computer languages, the limited conformity of usable specification languages (e.g. ASM) and the limited efficiency of automatically derived language tools (Mosses, 2006; Finkel, 1996).

In practice, dynamic semantics are typically implemented by language tools using one of two approaches - in absence of other specifications of dynamic semantics they are even defined by (Parr, 2009):

- *Interpretation* the language tool directly realizes the intended dynamic semantics while navigating a representation of concrete or abstract syntax.

- *Translation* the language tool translates abstract syntax into concrete syntax of another language (e.g. machine code, interpreter code).

In conclusion, the manual implementation of static and dynamic semantics and the lack of modularization cause concrete-syntax-oriented grammar-based languages to be rather defined as monoliths that are often built from scratch (Krahn et al., 2008). Against the background of the benefits of modularity and object-orientation in software engineering, the grammar-based, syntax-oriented specification of computer languages appears as having limited reusability of existing language artifacts and abstract patterns leading to relatively high costs associated with the development of computer languages, in particular when developing computer languages with relatively small application areas (Krahn et al., 2008) and when aligning/combining related languages that form a family of languages (Fischer et al., 2004). Metamodel-based approaches that are associated with the development of Domain-specific Languages try to overcome these limitations, by focusing on a graph-based specification of abstract syntax, instead of tree-based specification of concrete syntax as the base for the specification of computer languages (see Chapter 3.5).

3.2 Design of Computer Languages

The notion of "pragmatics" of computer languages is, compared to syntax and semantics, rather informal and diverse and there is no clear account to what it refers to, however it is commonly perceived as a fundamental aspect of the design and evaluation of computer

languages. Originating in the field of semiotics, pragmatics originally refers to the relation between sign and interpreter. In a slightly broader view, pragmatics refers to the relation of languages and the context of usage, including the general relation between users and language. This encompasses the efficiency of usage of language tools and program execution, the relation of application domain and language and the relation of user, language and computer (Zemanek, 1966). Thus, the functionalities of language tools that support users of computer languages are fundamental aspects of pragmatics: syntax highlighting, code completion, error detection, debugging and profiling.

However, the common notion of pragmatics is more comprehensive. Literature suggests a number of design and evaluation criteria of computer languages and respective language tools (Watt, 2004; Sebesta, 2004):

- Efficiency of tool implementation - depending associated number of applications, the costs of the development of language tools might prohibit their realization.
- Efficiency of program execution - ability to describe efficient computations, where execution time is limiting factor.
- Reliability - specifications behave as expected under any circumstances.
- Universality - language should provide the means to solve all problems of the intended problem domain.
- Writability - ease with which specifications are produced by humans.
- Readability - ease with which specification are read and understood by humans.

Whereas the efficiency of tool implementation is subject the approach used for developing language tools, efficiency of execution is subject to the design of language and language tools. Universality corresponds to the intended use. Writability and readability are indeed depend on how elements of the language match the skills of users, but also the characteristics of language tools with respect to development, e.g. related to debugging, profiling, deployment, testing are influential. The notion of regularity is commonly used as the framework for evaluation of pragmatics which is considered in Chapter 3.2.3 below. The following chapter sketches basic aspects of how computer languages are perceived to relate to application areas.

3.2.1 General-purpose Programming Languages and Domain-specific Languages

The history of computer languages shows that no optimal computer language exists and a number of computer languages have been designed, each providing a compromise between conflicting design goals.

For historical reasons, computer languages can be separated into *programming languages* and *modeling languages*. In principle, there is no distinction between the two, though historically, programming languages are typically rather associated with directly implementing executable computer programs and grammar-based specification languages. In contrast, modeling languages are rather related to the notion of specifying a particular aspect of a (software) system at a particular, relatively high, level of abstraction, not necessarily intended for execution, but also for communication, documentation and

reasoning. Further, programming languages are typically associated with textual concrete syntax, whereas modeling languages are rather associated with graphical concrete syntax. A similar distinction is made between *General-purpose Programming Languages* (GPL) and *Domain-specific Languages* (DSL), where GPLs are programming languages that are perceived as being applicable to a variety of application domains, whereas DSLs should provide domain-specific concepts. Taking into consideration that the development of any programming language is developed against the background of an intended area of application, this distinction is rather arbitrary. However, one might refer to two basic characteristics of computer languages to shed light on this distinction:

- *Expressive power* refers to the capability of a language to enable the use of the computational features of a real or imagined computer that is thought to execute a program (typically Turing-machine, where real computers are thought of being theoretically equivalent to a Turing-machine).

- *Expressivity* refers to the capability of a language to provide concepts that are rather aligned with the problem domain than with the target computer, in order to facilitate a relatively compact, convenient and intuitive formulation of programs.

Against this background, a computer language might be classified as GPL, when a key characteristic of the language is expressive power, thus existing computational features of the computing device should be available irrespective of the application domain (universality), whereas a DSL might sacrifice expressive power for the improvement of the expressivity. For simplicity, the terms "programming language", "modeling language" and "computer language" are used interchangeably in this text, since all computer languages are discussed against the background of M&S, including the implementation of executable simulators. The distinction between GPL and DSL is used in this text and further detailed in Chapter 3.3, since this is well-aligned with current approaches and issues of computer languages in M&S and EMS.

The adequacy of provided abstractions is a widely discussed issue in the field of programming languages. This discussion is typically framed into the discussion of programming language paradigms. The next section describes basic programming language paradigms and abstraction mechanisms that influence the design of programming languages, followed by the presentation of further common evaluation and design criteria of programming languages related to the notion of "regularity" of programming languages.

3.2.2 Abstraction and Programming Language Paradigms

The language concepts influence how programmers conceptualize a system (Watt, 2004). Thus they build the interface between humans and technology. Sets of key abstractions of programming languages are referred to as *programming language paradigms*.

The history of programming languages shows that abstraction typically refers to abstracting from machine instructions: Early low-level assembly languages directly provide machine instructions (e.g. direct memory access) allowing the formulation of efficient programs. Higher-level languages provide more abstract means to describe computations (e.g. variable assignment) and to organize complex descriptions (e.g. modularization). In general, programming language paradigms have been mainly influenced by the available hardware and the software development methods (Sebesta, 2004).

There exist a variety of programming language paradigms for GPL, which in many programming languages appear in combination (see Sebesta (2004) or Watt (2004)): *Imperative languages* describe computations in terms of ordered statements that change the state. Such statements are perceived as relatively compact and natural specifications of computations that involve a number of low-level directives of assembly or machine language. Efficiency of execution is typically associated with imperative programming. Typical concepts of high-level imperative languages are (typed) variables (e.g. characters, floating point, integer), statements (e.g. arithmetic, logical), loops (e.g. while, for, do), jumps (e.g. goto), and branching (e.g. if-then-else, switch).

In contrast, *declarative languages* are perceived as more natural in that they rather describe what to achieve and not which computations achieve a goal:

- *Functional programming languages* provide means to express programs as collections of mathematical functions, where in contrast to the imperative paradigm explicit state changes, thus side effects, are avoided (LISP, Scheme, Erlang, Haskell). Typical concepts related to the usage of functional languages are functions, higher-order functions (functions that have functions as arguments), recursion, currying, continuations, etc.

- *Logical programming languages* (e.g. PROLOG) define computations by means of logical inference rules that are applied to a knowledge base in order to infer the wanted result of the computation. Concepts used for programming are such from predicate and propositional calculus (predicates, propositions, logical constants, logical inference, backtracking etc.).

In practice, imperative style and declarative style are often mixed (Lisp, Erlang, Prolog, SQL).

The possibility of language users to introduce modularization and custom abstractions are besides efficiency perceived as basic properties of programming languages. Two basic kinds of abstraction play central role in the design of high-level programming languages: *process abstraction* and *data abstraction*. Process abstraction by means of functions and procedures allows programmers to introduce explicit grouping of statements (within functions and procedures), decomposition and reuse of groups of statements. Process abstraction particularly supports of the software engineering method of procedure-oriented structured programming in which the system to be developed is thought of being composed of procedures and functions that act on data (Sebesta, 2004). With increasing complexity of programs, data abstraction gained importance that focuses on structuring programs where operations and data are more closely related. Data abstraction mainly refers to the possibility to introduce abstract data types, where an abstract type is a set of values and a set of associated operations that exclusively operate on the set of values of the type (e.g. integers as the value set and "+" as an operation). Object-orientation provides state-of-the-art data abstraction with the possibility to define structured and nested types by the specification of classes in support of the method of Object-oriented Analysis and Design (see Chapter 2.2.3). Popular GPL combine object-orientation with the imperative or declarative programming paradigm (e.g. C++, Java, Scala, LISP/CLOS) supporting the method of Object-oriented Analysis and Design.

3.2.3 Criteria for the Design and Evaluation of Programming Languages

The design of computer languages may be framed in a discussion of *regularity*, which refers to the characteristic that a computer language can be used without the need recognize a number of exceptions of the rules of the language and without surprises with respect to behavior resulting from unanticipated interaction of language concepts. There is no common concrete definition of regularity, but literature suggests that regularity, and to a certain degree writability, readability and reliability, may result from adherence to the following interrelated criteria (see Watt (2004); Louden (2003); Sebesta (2004)).

- *Simplicity* - a small number of basic concepts and avoidance of multiplicity (multiple ways of expressing same structures and processes).
- *Orthogonality* - all combinations of basic concepts are legal and meaningful (e.g. types: integer, character; operators: array and pointer). The meaning of appearances of of orthogonal constructs is independent of their context.
- *Type completeness* - There are no exceptions with respect to the usage of types (all types are allowed where types appear).
- *Correspondence principle* - the usage of names is independent from the mechanism of introduction of a name (parameter/declaration).
- *Abstraction principle* and *qualification principle* - it is possible to introduce abstractions over any meaningful syntactic class and any meaningful syntactic class should allow local definitions.
- *Generality* - avoidance of special cases and combination of closely related concepts into a single one (e.g. operators should be equally applicable to types, e.g. exception: C: "==" not applicable to arrays).
- *Uniformity* - similar things look similar and different things look different.

Indeed these criteria are related, e.g. the correspondence principle is a way to enhance uniformity, the orthogonality and type completeness basically refer to the same concept and support generality. A computer language with a high degree of simplicity may require a high degree of generality. A high degree of simplicity however might lead to shortcomings related to readability and writability which may be dealt with by implementing the abstraction principle, thus the possibility to introduce new abstractions that conflict with simplicity. A high degree of generality might lead to issues related to uniformity since semantic differences (e.g. "==", equality of numbers and equality of objects) might be perceived differently and be hidden behind syntactic uniformity. Thus typical design criteria of computer languages that are associated with the goal to provide regularity appear to be related and conflicting. However, they indicate perceived shortcomings of programming languages that have been developed in the past. To be meaningful however, these criteria must be specified against the goals of a specific computer language with specific goals and user groups. Since the supported design goal of regularity is finally a characteristic that depends on practices into which the use of languages is embedded and the cognitive characteristics of users, the evaluation of regularity requires assumptions about users and their practices and cognitive-educational background.

The following chapters present existing tools, that reflect common technical issues and solutions, but also common practices and issues related to existing languages and language tools.

3.3 Tool Support for M&S

This chapter presents most influential approaches to tool design and implementation in M&S in general and the specific developments in the field of EMS. Due to the sheer number and diversity of modeling tools and computer languages used for M&S and EMS, a complete presentation of tools is beyond the scope of this thesis. Instead, after some general remarks, the following section presents a - knowingly incomplete - overview of most important characterizing features of available M&S technology at the base of exemplary tools and languages.

The most general characteristic of tools for M&S is that they provide abstractions by means of respective ready-to-use operational implementations that are useful across different M&S studies. Common M&S-specific abstractions can be grouped into abstractions that provide means to conceptualize:

- models as Dynamical Systems, including repeatable non-determinism by means of pseudo-random number generators and the consideration of concurrent state changes,
- properties of data input, output and visualization (e.g. format, access),
- properties of experiments or experiment series (e.g. parameter variation) and
- data analysis (e.g. aggregation).

There are generally two methods to provide specific abstractions: *internal DSLs* and *external DSLs*. Internal DSLs are defined by exploiting abstraction mechanisms of GPL (e.g. object-orientation).

> *An internal DSL is a DSL that uses syntax and semantics of a host computer language by using the host language's extension mechanism (e.g. data and process abstraction).*

In contrast, an external DSL is defined as a separate language.

> *An external DSL is a DSL with its own syntax and semantics.*

Whereas the creation of internal DSLs exploits existing language tools, the realization of external DSLs requires implementation of new language tools. However, whereas internal DSLs use the syntax and semantics of the host language, external DSLs may define their own syntax and semnatics.

Besides the universality of abstractions, it is the quality of algorithms that characterize specific tools for M&S and which motivate their development. Qualitative characteristics of tools are generally strongly related to the non-functional properties of the basic technology (e.g. hardware/software platforms) that is used to realize a modeling tool. Such basic technologies range from relatively simple standalone Personal Computers (single processor) to distributed computing platforms (e.g. component technologies such as

.NET, CORBA, Web Services) and from to parallel computing platforms (e.g. MPI) to cloud computing, with different characteristics related to simulation speed, reusability of models and efficiency of implementation.

3.3.1 General-purpose Programming Languages

General-purpose programming languages (GPL, see chapter 3.2.1), such as C/C++, FORTRAN and Java, are not specific to M&S, but they are nevertheless widely used for M&S and EMS. Depending on the programming language paradigm used, these languages used are characterized by the availability of relatively low-level language constructs tailored towards universality and abstraction mechanisms.

For many GPL there exist a variety of well-developed tools for programming, such as editors, compilers, debuggers, test suites for efficient implementation and execution of programs. GPL provide great degrees of freedom since they are universal with respect to underlying computing technology. Access to relevant new basic computing technologies (e.g. component technologies) is typically first provided by the means of existing GPL (e.g. through language tools, libraries etc.). Models implemented with GPL usually provide some means of reuse since it is possible to run code on a number of hardware platforms and implementations may leave the possibility to adapt the implementation of a model to an application by setting parameters, initial state, and simulator configurations without modification of code. Moreover, specific experimentation procedures (e.g. for calibration and data analysis) can be implemented and distributed as one reusable unit allowing for straightforward replication of studies. Built-in abstraction mechanisms allow for the provision of reusable and application- and domain-specific abstractions, that however have to be accessed at the base of the abstraction mechanisms of the GPL (e.g. functions, classes, modules).

A major issue related to the use of GPL is the limited efficiency of model implementation, when functionality for model execution, experimentation and data analysis is to be implemented from scratch. Moreover, the reuse of model specifications across application contexts, with different technological requirements and technologies (GPL, platforms) is usually perceived as inefficient (wrapping, tool integration, reimplementation, Argent (2004)). More fundamental issues of using GPL are caused by the conceptual gap between GPL and the problem domain[5]. Published reimplemention studies (Keller and Dungan, 1999; Axelrod, 2003) show that, due to this conceptual gap it is typically not possible to fully understand the model based on code, since it is not possible to reconstruct the modeling concepts from it (Keller and Dungan, 1999). Further, it is typically not possible to reconstruct models from documentation without studying the GPL code, due to ambiguities in natural language documentation (Axelrod, 2003) [6]. Thus, the comprehension of a model implemented with GPL requires technical skill, intuition and experience, since

[5] The 'conceptual gap' refers to the fact that the language used for model specification does not provide means to directly represent those abstractions that are used in models cognitively (Villa, 2001). The conceptual gap appears such that several statements of the GPL are used to express a single specific modeling concept (Keller and Dungan, 1999) and statements related to modeling concepts are mixed with statements related to computational aspects, e.g. data management (Holzworth et al., 2008).

[6] Axelrod (2003) points particularly on the ambiguity of descriptive attributes, such as "artificial society, complex system, agent-based model, multi-agent model, individual-based model, bottom-up model, adaptive system" used in scientific discourse, but which are finally specified within the source code of the simulation model and where small differences in interpretation might cause substantial differences in behavior and interpretation.

comprehension requires knowledge about domain, GPL and individual programming style (Keller and Dungan, 1999; Axelrod, 2003). Issues that directly follow from limited comprehensiblity are limited replicability of experiments, limited transparency of M&S studies, thus flaws in communication and discourse and limited reuse of models (Muetzelfeldt and Massheder, 2003; Keller and Dungan, 1999; Axelrod, 2003; Fall and Fall, 2001).

3.3.2 Mathematical Packages

The numerical analysis of dynamical systems, particularly Differential (Algebraic-) Equations and Difference Equations has lead to a substantial body of theoretically well-defined concepts and algorithms for their simulation and statistical characterization (Zeigler et al., 2000). There is a great number of tools that provides implementations of these algorithms. Particularly relevant for M&S in general are numerical algorithms for numerical integration, pseudo-random number generation and statistical aggregation. Although mathematical packages share the theoretical mathematical background, remarkable differences with respect to amount of functionality, computer languages and user interface exist. Typically mathematical packages provide their own GPL or extend existing GPL with efficient data structures on which functions, such as integration of differential equations in a given time span, can be called. Influential exemplary developments with a relatively great amount of functionality are the Gnu Scientific Library (C/C++, API, Galassi et al. (2011)) and MATLAB (MAT, 2012), Gnu Octave (Eaton et al., 2011)) and Modelica (Fritzson, 2003).

Mathematical packages may offer specific tool support for M&S studies (e.g. experimentation, visualization) and there exist mechanisms to integrate functionality written in other programming languages and vice versa (Galassi et al., 2011; Fritzson, 2003; MAT, 2012). However, functionality and expressive means are perceived as general because modeling language is GPL (e.g. GSL) or oriented towards general mathematical concepts (vectors, matrices, equations and operations that perform on these), although abstraction mechanisms can be used to provide specific abstractions. Further, more universal discrete-event modeling and explicit compositional modeling is typically limited (except Modelica that is designed to express models compositionally). Although these tools provide verified implementations of well-defined efficient simulation algorithms they typically encompass the advantages and issues related to GPL, although relative efficiency and transparency is evident in the case of Differential (Algebraic-) Equations and Difference Equations, compared to GPL. Issues are related to concentration towards equation-based modeling in combination with the generality of GPL, thus comprehensibility, and reusability of model specifications across applications (see Chapter 3.3.1).

3.3.3 Tools for Combined Modeling

A number of tools provide discrete-event modeling and combined modeling. In contrast to mathematical packages, where theory preceded tool development, the discrete-event formalisms, thus the formal mathematical semantics of respective languages, followed the development of tools (Zeigler et al., 2000). Thus, there are notable conceptual and technical differences (see below), but combined discrete-event modeling is commonly recognized as providing a universal time-advance and synchronization mechanism for combined compositional digital simulation models (see for examples Zeigler et al. (2000); v. Evert et al.

(2005); Vangheluwe et al. (2002); Praehofer et al. (1993))[7]. Zeigler et al. (2000) illustrates this at the example of the family of DEVS-like formalisms and a universal simulation algorithm (DEVS-Bus) that enables the combination and simulation of any discrete-event and combined continuous and discrete event formalisms as long as there exists an integration procedure that approximates continuous behavior with an arbitrarily small error (such as provided by mathematical packages).

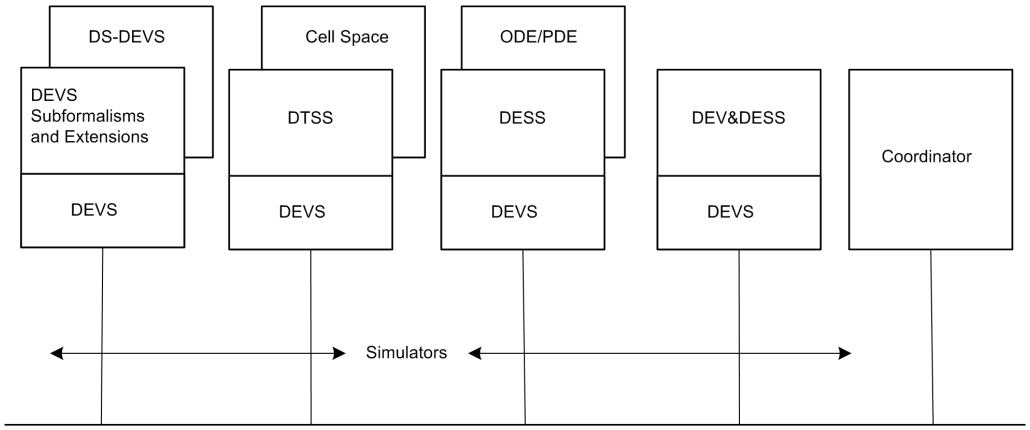

Figure 3.1: The DEVS-Bus enables simulation of arbitrary compositional combined discrete-event continuous models by means of hierarchical ordering of coordinators.

The functional core of these tools is a simulation algorithm that processes events in a predefined order, primarily based on time ordering and in case of concurrent events, a further ordering is given explicitly, by priority or explicit random ordering. Events[8] are put into the event calendar according to their time (and priority). For this, events are scheduled explicitly (time event) or prescribed by a condition that triggers an event (state event). In combined simulation, event detection is based on numerical integration of differential equations. Although this time flow mechanism is general, the efficiency of simulation of different discrete-event approaches varies significantly depending on the structure and variability of the structure of models. Thus although theoretically general, practical simulation efficiency limits real applicability to specific implementations (e.g. variable structure may induce coordination overheads that prohibit bus-based solutions, or coupling continuous systems might require an integrated numerical procedure). This might explain the great number of variants and different implementations (see below). The discussion of simulator efficiency however is beyond the scope of this thesis.

The universality of the discrete-event simulation procedure and the possibility of compositional modeling is naturally combined with object-orientation (Zeigler et al., 2000) that supports both, structural decomposition and provision of specific abstractions (e.g. paradigms) through internal DSLs. Thus in practice, modeling is basically realized by specialization of given templates (e.g. classes), through setting and adding attributes, and adding of functions and implementing of virtual functions, e.g. the state transition

[7] Universality is here considered irrespective of constraints related to the efficiency of execution, but only with respect to expressive power.

[8] In case of process-oriented worldview, it is a reference to a process that is put into the event calendar, which proceeds its lifecylce at the time of the event.

function, which in external DSLs may appear as a separate language construct. Behavior is basically defined by means of event routines (event-oriented view) or lifecycle routines (process-oriented view) and event conditions, all defined according to a template (e.g. superclass, grammar). With internal DSLs the low-level means of the host GPL are used for the concrete specification of these routines. Thus, there is considerable degree of generality and universality of available language constructs, such that general issues apply at an abstraction level below the required interface.

Moreover, there is notable heterogeneity in tools. In general, tools differ with respect to world-view, but even within a particular world view, the description of aspects such as model coupling (e.g. mere pointers, messages, output/input functions, data format) and syntax differ, leading to a relative heterogeneity of perceptions and concepts. An illustrative and well-documented example illustrating the variability of discrete-event simulation is the family of DEVS-based modeling formalisms and tools, that is perceived as a theoretically relatively well-founded discrete-event formalism. The *Discrete-Event Specification System* (DEVS) is a formalism for the specification of event-oriented models of which Zeigler et al. (2000) presents 9 variants and extensions[9]. DEVS simulation algorithms have been developed for serial, parallel computation (shared memory) and distributed computation (distributed memory). A number of tools have been developed that support DEVS-based modeling and simulation[10].

The field of tools for process-oriented modeling shares these characteristics. There are numerous tools available with a number of variants for internal DSLs (e.g. jDISCO see Helsgaun (2001), ODEMx see Fischer and Ahrens (1996), C++/CSIM, C++SIM see Joines and Roberts (1998)) and external DSLs (GPSS, SIM-SCRIPT, SIMULA) in combination with object-orientation or procedural style and a variety of coupling styles (Joines and Roberts, 1998). Although process-oriented tools provide a simulation procedure that can be used to model discrete-event models[11], there is a considerable amount of heterogeneity and generality of modeling concepts.

However, tools for discrete-event modeling and combined discrete-event modeling provide verified implementations of universal simulation procedures for compositional modeling for a variety of computing platforms with different non-functional properties. Although object-orientation provides means to provide different levels of abstraction, issues of GPL apply nonetheless due to generality of the event-based time-flow and synchronization mechanism, the expressive means to specify structure and transitions, and technical and conceptual heterogeneity of available tools.

[9] Classic-DEVS as the most simple formalism, parallel DEVS for explicitly expressing parallelism in the model (P-DEVS), discrete event specified network formalism (DEVN) for explicit modeling of couplings with variants for P-DEVS and classic DEVS, multicomponent DEVS (multiDEVS) for explicit compositional modeling, discrete-event and differential equations specified system (DEV&DESS) for combine discrete-event and continuous modeling, dynamic structure DEVS for explicitly modeling structural changes (DSDEVS), symbolic DEVS for dealing with a number of trajectories simultaneously, real time DEVS for modeling with real time, fuzzy DEVS for modeling with uncertainty.

[10] Examples of DEVS-based tools: e.g. adevs / Nutaro (2011), DEVS++ / Hwang (2009)), C# (Hwang (2007)), Java (DEVSJAVA / Zeigler and Sarjoughian (2005)), Common LISP/CLOS (STIMS-CLOS) and other languages implementing different DEVS-variants and specializations (for an overview: `http://en.wikipedia.org/wiki/DEVS#DEVS_Tools`).

[11] However, the implementation of efficient process-oriented simulation kernels is relatively demanding (see Kunert (2010))

3.3.4 Component-based M&S

The aim to reuse model specifications across diverse technologies and application contexts motivates developments that follow the idea of federated simulation, usually associated with the notion of component-based M&S[12]. The basic idea of federated component-based modeling is that single models can be technically reused in different contexts using a component-based computing platform that provides basic services to coordinate simulation (synchronization and data exchange), where single components typically implement some model-specific simulation functionality and which are loosely coupled at runtime for co-simulation. Federated simulation naturally lends itself to distributed computing, where different components reside on different connected computing devices. Single models are conceptualized and implemented as *model components* using any computer language compliant with the component platform. As an independent unit, a component only interacts with other components and the component platform via a predefined interface. Required interfaces for models might be such that meta-information must be provided by model components enabling the platform to provide services for model management supporting the specification of federated simulations with a number of interacting models. Details of implementation (e.g. programming language, hardware) are hidden, allowing the cosimulation of models based on different technologies and the exchange of implementations (e.g. HLA (Department of Defense High-Level Architecture) is an prototypical effort that prescribes interfaces and an architecture that has been implemented by number of tools (Dahmann et al., 1997)).

The intricacies of of conducting M&S experiments that require the integration of numerous models, analysis tools and large amounts of data against the background of the rate at which basic technologies change, motivates the development of *Scientific Workflow* technologies (Ludäscher et al., 2009). Scientific workflows aim at automation of tasks of experimentation with digital simulation models, the exploitation of latest technologies and the documentation of experiments, experiment series and data for replication of experiments (Ludäscher et al., 2009). Experiments are perceived and defined as an ordered execution of tasks, where tasks (e.g. data preprocessing, simulation) are dependent on former steps (data exchange) and executed by means of possibly distributed computational resources (Ludäscher et al., 2009). Models are to be designed as components conforming to the requirements of scientific workflow tools, that besides the mentioned general characteristics have remarkable conceptual and syntactical differences (Ludäscher et al., 2009; Hardebolle and Boulanger, 2009). For designing workflows there are usually visual languages provided by tools (e.g. Kepler (Kep, 2012), Taverna (tav, 2012), Triana (tri, 2012)) that manage workflow definitions, workflow execution and data (Ludäscher et al., 2009).

The component-based approach allows the technical integration and technical reuse of models across basic computing technologies and applications and - in the case of scientific workflows - a relatively high degree of documentation of experiments, models and data. However, issues remain due to the use of GPL for the specification of basic model components and heterogeneity of languages for model integration. Further, developments highlight the need to deal with technically demanding experimentation processes where the separate description of experimentation procedures are is seen as an approach to address this complexity. A distinguishing characteristic of components is that they are

[12]Development here indeed parallels and is based on the development of component technologies such as (CORBA, COM, J2EE and in a wide sense Web-Services).

usually bound to a specific modeling tool, by the need to conform to a specific interface. Further, the support for reuse primarily addresses the issue of bridging technologies and institutional and spatial distances. Much of feasibility has to be established by modelers, e.g. that operational semantics fit and validity is ensured. A further issue is that general component frameworks come along a coordination and communication effort, which is particularly present in case of distributed computing with model components connected via local or wide area network.

3.3.5 Domain-specific Languages and Multi-paradigm Modeling

The issues related to the generality of GPL have been addressed by tools that provide DSLs. A number of GPL-based tools provide theoretically universal synchronization mechanisms and ready-to-use basic simulation algorithms and functions (e.g. combined discrete-event (Chapter 3.3.3), component-based tools (Chapter 3.3.4), hybrid automata (Hardebolle and Boulanger, 2009)). Based on this powerful simulation functionality, abstraction mechanisms support the provision of distinct sets of abstractions, e.g. paradigm-specific DSLs for *multi-paradigm modeling*. However, issues related to the use of GPL (comprehensibility, transparency) and heterogeneity (reuse) of tools remain in case of internal DSLs.

> *Multi-paradigm modeling is an approach to M&S, where several DSLs that incorporate different modeling paradigms are used for model specification.*

This definition subsumes approaches that are referred to as "multi-modeling" and "multi-formalism modeling" in literature as it refers to models that are structured along the perceived structure of a system, organizational structures ("views") or levels of abstraction.

There is a variety of tools that provide single external DSLs for system modeling and efficient experimentation and simulation (e.g. SIMULINK). Typically issues of reusability apply due to the monolithic character of such tools. The provision of multi-paradigm modeling with several external DSLs that are used in combination with reusable models that are not bound to a specific tool is a present research issue, as presented in the following chapter.

Multi-paradigm Modeling with External DSLs

For many general issues that are related to the combination of models and which apply to any sort of multi-paradigm modeling (e.g. synchronisation, semantics etc.), there is remarkable theoretical and practical background associated with combined modeling and component-based modeling. The specific issue related to multi-paradigm modeling with external DSLs is the efficient integration of external DSLs within modeling tools[13].

Tools that integrate several external DSLs typically follow one of the following strategies (see Hardebolle and Boulanger (2009)):

- Joint use of modeling tools - existing tools and their DSLs are used separately and are integrated based on co-simulation, where the simulators of tools interact at runtime (e.g. MUSIC: SDL-Matlab). Technologies for integrative M&S can be used for integration if simulators, e.g. by means of wrapping (e.g. HLA, SystemC).

[13] Language tools that provide computer languages for experimentation and simulation modeling are further referred to as "modeling tools".

- Composition of DSLs - a tool supports several interrelated DSLs. Complex models can be analyzed at several levels of abstraction, from the level of specification to the level of simulation. This approach may be based on a language the subsumes all used DSLs (super-formalism, d. Lara and Vangheluwe (2002)), it may involve the transformation of different models into a common language and co-simulation (d. Lara and Vangheluwe, 2002).

The joint use of modeling tools allows the exploitation of existing possibly optimized implementations of editors and simulation algorithms. Its main drawback is the limited scalability and reuse, assuming that different applications of a model requires different sets of DSLs causing the need to manage tool integration with increasing complexity (Hardebolle and Boulanger, 2009). Moreover, characteristics of the joint models are available at the level of the simulation technology only, thus higher-level information is not available in the higher-level modeling tools for analysis of joint characteristics. In contrast, the composition of DSLs in one tool allows for comprehensive analysis (and enforcement) of properties of combined models within the modeling tool. Tools with several combined DSLs however have to implement the functionality for specification and execution of all DSLs (syntax and semantics), instead of relying on existing language support which may lead to prohibitive implementation costs, since for every new combination of DSLs, langauge tools have to be implemented. High-level language description techniques (such as formal grammars and language meta-modeling, see chapter 3.1.2) promise relatively efficient realization of external DSLs compared to ad-hoc implementation. The approach of *Computer Automated Multi-paradigm Modeling* (CaMPaM) and *ATOM*^{3}, a research tool for M&S that implements CaMPaM, exemplifies efforts that are focused on combined external DSLs in combination with efficient language implementation.

Computer Automated Multi-paradigm Modeling and ATOM^3

Mosterman and Vangheluwe (2004) defines *Computer Automated Multi-paradigm Modeling* (CAMPaM) as a framework that addresses M&S with respect to three dimensions: first, the coexistence of models of the same system at several levels of abstractions and, second, the concurrent use of several related DSLs for the specification of models. Third, inspired by model-driven approaches in software engineering, CAMPaM puts language metamodeling and the promised relatively efficient implementation of DSLs in the focus of tool development for M&S (language metamodeling is presented in Chapter 3.5 in more detail).

The approach to deal with the first two dimensions (different levels of abstraction and several DSLs) is based on the transformation of models between DSLs and can be illustrated at the example of the *Formalism Transformation Graph* (FTG) as presented in Vangheluwe et al. (2002)[14]: A number of languages (formalisms[15]) for the description of dynamical systems and relations between them are depicted in the FTG (i.e. *PDE*, *System Dynamics*). For any language in the FTG there exists a simulator, which is denoted by dotted lines to *state-trajectory data*. Other arrows (solid, dashed) denote that models in one language "can be mapped onto" a model in another language by means of symbolic transformation. It might be possible translate models of a language into a

[14]The third dimension - language metamodeling - will be presented in a more general context in Chapter 3.5

[15]The author uses the term formalism to to refer to the fact that a formalism is a computer language with formal syntax and semantics.

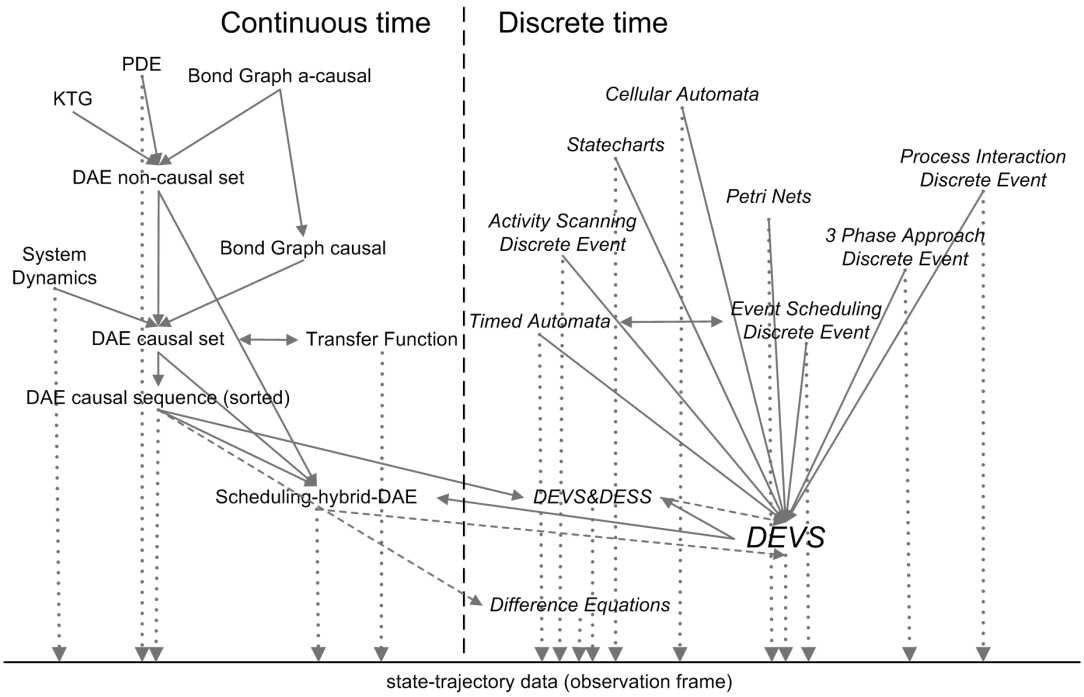

Figure 3.2: The Formalism Transformation Graph (from Vangheluwe et al. (2002), modified)

model of another language without loss of precision with respect to observed state (denoted by solid arrow). Further, it might be possible to approximate models written in one language by a model of another language with accepted accuracy (dashed arrows). Such transformation-based approach allows to give semantics to new languages by provision of a mapping to an existing language. Further it allows to couple models and analyze properties of the joint model, when models of different languages are transformed to a model in a common language. Different models according to different languages allow for different kinds of analysis for answering different questions, in particular the fulfillment of non-functional constraints (e.g. verification of properties, model checking). Finally, transformations lead to a representation in a language that represents a "common denominator" in the sense that it allows the co-simulation of coupled models (Vangheluwe, 2001). Indeed this common denominator must obtain relative universality. The FTG denotes the DEVS formalism as a common denominator, but others, e.g. Ptolemy and S-functions, have served this function (see Mosterman and Vangheluwe (2004)) and it is assumed that other powerful languages following a universal paradigm (e.g. combined process-based or hybrid automata) may serve the same purpose.

The transformation-based approach based on the FTG has partly been implemented in a tool for spatial modeling and simulation of wastewater-treatment (WEST++) that however does not implement metamodel based language definition, but includes a superformalism (MSL-USER) for modeling with System Dynamics and DAE that committed to the development of Modelica and uses transformations to analyze models (e.g. algebraic loop detection) and derive simulators (Vangheluwe, 2001). ATOM^3 is a research-oriented tool that implements CAMPaM with meta-models that has been used to implement the DEVS formalism, Petri nets and Statecharts, GPSS, causal block diagrams, and flow

diagrams and has been particularly applied in the domain of embedded control systems (Mosterman and Vangheluwe, 2004). In general, CaMPaM and the FTG present a realization of paradigm specialization and paradigm approximation (see Chapter 2.3.3) by means of transformation between computer languages for the sake of efficiently realizing multi-paradigm modeling with external DSLs.

3.4 Tool Support for EMS

The domain of EMS shares a number of issues with M&S in engineering (e.g reuse, implementation and execution efficiency) and many approaches to tool support presented above have been applied to EMS. However, some specific characteristics can be identified at the first sight: First, many environmental issues require the explicit consideration of geospatial dimension, thus geospatial data processing and analysis (Argent, 2004). Second, the high degree of uncertainty leads to a relatively great degree of adaption and modification of existing models which leads to high requirements with respect to transparency for scientific soundness (Voinov et al., 2004). Third, EMS is dominated by the goal of system inference at lower levels and system analysis (prediction, scenario analysis) at higher levels of the model lifecycle. Fourth, modelers in environmental science do not have an explicit technical educational background, such as modelers with a background in engineering.

These specific characteristics may motivate the development of the great number of specific tools for EMS, which however are usually applications and extensions of techniques developed in systems and software engineering (see above). This chapter aims at elaborating the specific characteristics of EMS in contrast to M&S in engineering at the base of characteristics of existing tool and approaches. First, some general notions from the scientific discourse about tools and tool design for EMS are presented (Chapter 3.4.1), followed by general remarks on the widespread application of object-orientation in EMS (Chapter 3.4.2). The following chapters (Chapters 3.4.3, 3.4.4 and 3.4.5) present current main approaches to tool design in EMS. Chapter 3.5 presents the approach of Model-driven Engineering, the application of which to EMS and its evaluation is subject of this thesis. Chapter 3.6 concludes Chapters 2 and 3 by relating basic characteristics of EMS and MDE.

3.4.1 Basic Classes of Modeling Tools in EMS

Scientific discourse about tool support for EMS shows that the level of abstraction provided by the modeling tools is subject to discussion. Rizzoli et al. (2005) and v. Evert et al. (2005) distinguish "implementation-level" tools for the specification of single models at lower levels of model lifecycle and "modeling-level" tools that support management and integration of different models for "integrated modeling" at later stages of model lifecycle. Whereas implementation-level tools are associated with the notion of "modeling frameworks" that allow the specification of single models as composable "model components", the modeling level tools are associated with the notion of "modeling environments" that provide application-specific high-level user interfaces for the combination of model components and experimentation support (Argent, 2004). "Integrated modeling frameworks" should allow the specification of models according to different paradigms and domains (Rizzoli et al., 2005) and may be designed as modeling frameworks such that they build the technological basis for the development of modeling environments for integrated modeling (Argent, 2004).

In addition, modeling tools are explicitly classified according to the level of abstraction they provide for the specification of single models. Fall and Fall (2001) identifies three classes of tools: first there are universal modeling tools that basically provide GPL as modeling language, second there are tools that allow model specification according to higher-level modeling paradigms (e.g. Petri nets, Cellular Automata), third there are tools that provide means to specify models by setting parameters of "pre-programmed" models, thus parameterizable models.

A further basic distinction is made between "imperative modeling languages" that provide means to express models as a series of computations and "declarative modeling languages". However, the notion of "declarative modeling language" is not clearly defined, but rather stated by means of design goals: These "declarative modeling languages" should enable modelers to describe models in form of mathematical statements the bare relatively "closer resemblance to the way environmental scientists conceptualize systems (Villa et al., 2009)." "Declarative representation makes models far easier to re-use and combine, and much more transparent than implementations within a traditional, code-based, model structure (Rizzoli et al., 2005)." Further, declarative modeling languages should enable modelers to implement simulators with less effort and technical skill (Muetzelfeldt and Massheder, 2003)[16]. Further, declarative modeling is increasingly understood as a means to separate model specifications and aspects of simulation technology and thus, facilitate the use of model descriptions with different simulation technologies in different application contexts (Villa et al., 2006). However, "declarative languages" have been recognized as difficult to realize since respective language tools have to implement full language support and different tools have to agree on common on declarative languages for reusability (Argent and Rizzoli, 2004). Due to equivalence of design goals, what is usually referred to as "declarative (modeling) language" in the context of EMS is further referred to as "DSL" in this text and general considerations about computer languages make the distinction between "imperative" and "declarative" arbitrary (see Chapter 3.2).

3.4.2 Object-orientation and EMS

As object-orientation can be seen as the "chief" amongst the software engineering approaches that underly technological progress in tool support for EMS (Argent, 2004), there are some general remarks on the application of object-orientation in EMS: At the first place, object-orientation provides a powerful abstraction facility to provide internal DSLs and to provide reusable implementations of M&S functionality. Moreover, object-orientation widely perceived as providing an relatively intuitive natural way to represent structures of systems compared to other abstraction mechanisms of programming languages (e.g. functions, procedures, see Silvert (1993)). Further, adequacy with respect to reuse, decomposition, implementation efficiency in comparison to procedural programming languages, such as FORTRAN (Silvert (1993); Keller and Dungan (1999)), is documented.

Main objections to the application of object-orientation to EMS are that in practice modelers tend to mix problem-oriented concepts with implementation-oriented concepts, such as process queues, lists, accumulators, grids, etc. " [...] that are irrelevant to the core modeling problem, but procedurally necessary for its computer solution (Keller and Dungan, 1999)". Further, although structural or conceptual ordering of aspects may be represented naturally (e.g. taxonomies, individuals), complex interaction between elements of models

[16] Please note that the term "declarative language" refers to computer languages at a relatively higher level of abstraction compared to "declarative programming languages" (see Chapter 3.2.2).

often leads to the specification of additional classes with no representative function, with a variety of ways these are conceptualized and modeling interaction appropriately requires modelers to have sufficient training in using object-orientation (Derry, 1998). Further, it has been observed that most object-oriented model implementations have been developed as monolithic models, where "everything depends on everything": model, data, numerical integration, optimization, calibration, display (Rizzoli et al., 2008a). Thus, in practice the merits of object-orientation are rather related to easing technical aspects of simulator implementation and code management, than to aspects of representation of the perceived ontological structure of models with respect to systems under study.

3.4.3 Geographic Information Systems (GIS) and EMS

A plethora of software provides GIS functionality. Although there is a historic separation of those GIS tools that are tailored towards processing of vector data (points, lines, polygons) and those tools for processing raster data (e.g. remote sensing data), it is state-of-the-art of comprehensive tools to integrate both. Comprehensive GIS tools (e.g. ArcMap (Arc (2012)), GRASS-GIS (GRA (2012)), ERDAS (ERD (2012))) are usually built upon implementations of core GIS functionality (see Chapter 2.1), which can be used and extended via API. In addition, Graphical User Interfaces provide access to typical GIS functionality. The API can be used to extend and customize GIS functionality or to implement specific stand-alone GIS tools. Further, it is possible to include GIS functionality in other separate applications (e.g. simulations) using common GPL and API. There exist a number of implementations of basic GIS functionality as shared libraries that can be used to implement tools, many of them open source (see (Ope, 2012) for a listing).

Many modeling tools and respective modeling languages for EMS do not include any facilities for explicit geospatial modeling (e.g. Stella, MATLAB). Coupling to GIS tools is realized by manually importing and exporting spatial datasets (Pullar, 2004). Many tools that include support for geospatial modeling evolved as modeling extensions of GIS tools (ESRI Modelbuilder, ERDAS Spatial Modeler, (Mazzoleni et al., 2006)) or evolved as spatial extensions of existing modeling tools for M&S (Stella, Modelmaker, Vensim, Extend and SIMILE). Further, many tools are custombuilt tools, where GIS and M&S are highly integrated (PCRaster, SME, IMA, Pullar (2004)).

The degree of integration of GIS functionality ranges from usage of I/O-related operations (retrieve, store, visualize), data aggregation (calculate statistical measures) to the application of GIS operations for modeling structure and dynamics of models (e.g. networks, local convolution operators, see Maxwell and Costanza (1997)).

Thus, in general GIS is involved in either pre-processing and post-processing of simulations, during simulations or all (Argent, 2004). Besides API-based integration and manual export and import of datasets GIS functionalities can be provided by GIS components that can be integrated with model components by means of respective component integration technology (e.g. Workflows and Web Services (Theisselmann et al., 2009b), Workflows with ESRI Modelbuilder (Mod, 2012)). Further the semantics of language elements of DSLs may be given by the means of GIS, such as spatial averaging or convolution (Maxwell and Costanza, 1997). GIS and M&S is "conceptually loosely coupled" when computations related to GIS and computations related to simulation of a dynamical system are conceptualized separately and conceptually connected via data input and data output. Such conceptual loose coupling may indeed be implemented by means of technical

loose coupling (e.g. manual or automated data exchange between processes) and technical tight coupling (e.g. API-based integration with shared memory). In contrast, "conceptual tight coupling" of GIS and M&S may refer to GIS-M&S integration where structural and behavioral aspects of models are conceptualized using concepts of GIS (e.g. convolution, spatial network etc.) or vice versa (e.g. agent-based map generalization, Duchêne and Gaffuri (2008)). Such conceptual tight coupling may be implemented by loose and tight coupling, however obvious considerations of computing efficiency (e.g. high frequency of data exchange) and modeling efficiency (complicatedness of specification) propose technical tight coupling in this case, because of its realitve computing efficiency and compactness of representation.

In general, the usage of well-defined GIS concepts and operations and verified efficient implementations can be regarded as serving modeling efficiency and credibility. However, the integration of GIS with M&S adds to the general issue of heterogeneity of technologies, comprehensibility of specifications and limited reuse of models, in particular when conceptual tight coupling is realized and GIS concepts are part of the conceptualization of models.

3.4.4 Component-based EMS

In EMS the term "component" is used in a wide sense. As a component that represents a model (model component) it is perceived as a software unit that is conceptualized and implemented such that it is reusable in many applications. Therefore it should have technical dependencies only to the modeling framework not to other model components. Model components are linkable such that several model components that represent interacting submodels can be executed in parallel as specified outside the model component (Rizzoli et al., 2008b).

According to this broad conception of the notion of component, there are several ways of realizing component-based modeling. There is a number tools for component-based EMS that exploit component and component-like technologies (e.g. TIME based on .NET (Rahman et al., 2003), ModCom based on COM (Hillyer et al., 2003), DIAS based on CORBA (Hummel and Christiansen, 2002)) that allow for integrating several GPL. However, many tools implement "ad-hoc" component mechanisms where model components are programmed by means of GPL. Model component specifications have to implement particular functions, a particular interface or derive from particular base classes (e.g. JAMS/OMS based on Java (Kralisch and Krause, 2006; Kralisch et al., 2005; David et al., 2004), Tarsier based on Borland C++ (Watson and Rahman, 2004), MMS based on C and FORTRAN (Leavesley et al., 1996b,a), ICMS based on MickL (Rahman et al., 2004b)). Components - the actual entities accessed by the modeling framework for setting up the simulator at runtime - might be code (e.g. MMS), compiled libraries (e.g. Windows DLL: OpenMI, Tarsier) or components (e.g. DIAS/CORBA, TIME/.NET, ModCom/COM, OMS/Java NetBeans).

Typically, model specification requires the implementation of functions for the initialization and destruction of components. Further a function that specifies the state change in a single time step is required (see Argent and Rizzoli (2004) for a comparison of different interfaces). Model components are typically connected such that inputs and outputs of different model components are connected. The models' transition functions are called according to the order of dependencies defined by the network of connections. However, remarkable differences exist with respect to the realization of couplings: whereas in

OpenMI models require inputs at times of execution (pull-mechanism), Tarsier implements the observer pattern, where modules get new input when the state of an input-component changes. ICMS employs a central scheduling mechanism, whereas MMS executes all models serially in a loop. There are different mechanisms to resolve circular dependencies (feedback loops) between model components: ranging from no treatment (MMS), to manual insertions of delay components to automatic treatment. Model components are typically connected based on the base of regular or irregular discrete time steps, continuous couplings are typically not supported. The realization of data exchange is approached differently and ranges from accessing a common local database (e.g. MMS) to message passing in a distributed network (e.g. DIAS).

For model coupling and setup of model components, there is usually a user interface or a tool-specific graphical or textual language available or tools support the development of such: MMS, OpenMI, ICMS provide their own graphical user interfaces and JAMS uses XML (MBUILD, Leavesley et al. (1996b)). A particular characteristic of component-based modeling frameworks is the integration of meta-data that allows users to specify information that a modeling tool based on the framework can exploit for supporting specification of coupled models. Although there is no standard set of information, meta-data typically encompasses information about parameters and state variables, input and outputs (name, domain, units etc., Rizzoli et al. (2008a)). Such information can be exploited to provide model component management, evaluation of couplings and automatic user interface generation for model components and analysis of (e.g. TIME, OpenMI, ICMS). The actual way this meta-data is specified varies in practice. With MMS and Tarsier variables and parameters are registered by means of implementation of a required function. TIME uses meta-data tags of the .NET framework to embed such information, APSIM (Holzworth and Huth, 2009) a combination of XML and .NET meta-data tags.

The integration of GIS ranges from linkage to GIS based tools for pre- and post-processing via database (MMS, GRASS, Weasel; Leavesley et al. (1996b)) over the provision of special datatypes such as raster, time series, node link network and point / line / polygon, Node Link Networks, Cross Sections with specific operations, to the provision of predefined GIS components (e.g. terrain analysis of rasters) that are loosely coupled to model components (e.g. Tarsier, TIME; Rahman et al. (2004a, 2003)).

The application of the notion of component-based EMS and federated simulation is considered to foster reuse and integration of models throughout the lifecycle of models, since tools for component based EMS serve as modeling frameworks at the base of which modeling environments can be implemented efficiently. However, reuse is typically bound to a specific component-based modeling framework, due to conceptual and syntactical differences between them. Further, the use of GPL and conceptually and technically complicated component technologies gives rise to issues of transparency and requires a considerable amount of software engineering skills (Watson and Rahman, 2004). Moreover, the more the technical issues of model reuse and integration are solved, semantic issues come to foreground, which can be supported by means of integrating higher-level information (e.g. units, application context, constraints), e.g. through metadata (XML, introspection, e.g. Holzworth and Huth (2009)) that can be exploited by the tool, e.g. by means of ontologies (e.g. IMA, see chapter 3.4.5). However, typically discussion and convention have to precede technical solutions for semantic issues (Argent, 2004; Argent and Rizzoli, 2004).

3.4.5 Integrated Modeling with External DSLs

A number of tools provide a single or several external DSLs for EMS. By means of a super-formalism (see Chapter 3.3.5), *Simile* combines System Dynamics with spatial modeling, object-orientation and individual-based modeling with the intention to provide a level of abstraction between GPL and paradigm-specific modeling (e.g. cellular automata, Muetzelfeldt and Massheder (2003)). Similarly, the *Spatial Modeling Environment* (SME, Maxwell and Costanza (1997)) supports the object-oriented *Modular Modeling Language* (MML) which is designed as a rather universal modeling language for environmental models. Models built with the Systems Dynamics language *Stella* (Costanza and Voinov, 2001) can be imported into SME. DAE-based DSLs used for environmental modeling are *Modelica* and *MSL-User* (Fritzson, 2003; Vangheluwe, 2001; Villa et al., 2009). The *Virtual Laboratory Environment* (VLE, Ramat and Preux (2003); Quesnel et al. (2009)) provides a number of external DSLs based on a object-oriented simulation kernel that implements simulators for different variants of DEVS (see chapter 3.3.3). This includes spatial modeling with cellular automata. Model templates are provided via subclassing and interfaces for modeling are implemented manually. Spatial configuration and connections between submodels can be edited graphically.

The presented tools exemplify efforts of modeling with several external DSLs in the fields of EMS. However, the presented tools share the characteristic that DSLs are either relatively general, bound to the specific tool or that DSLs are integrated via import functionality only. Approaches that address the issue of implementing modeling tools with coupled DSLs efficiently, thus with explicit consideration of flexibility at the base of combining DSLs, are rare. A notable, because comprehensive, example is the *Integrated Modelling Toolkit* (IMT), which is an implementation of an approach called the *Integrated Modeling Architecture* that integrates external DSLs and semantic processing.

The Integrated Modeling Architecture and the Integrated Modeling Toolkit

The *Integrated Modelling Architecture* (IMA, Villa (2007)) defines an approach to realize modeling tools for integrated modeling with several DSLs, where models are instances of formal ontologies. Thus, datasets and models "embody a statement of the system conceptualization, enabling machine reasoning about the system structure that can lead to more sophisticated applications (Villa et al., 2009)."

The *Integrated Modeling Toolkit* (IMT) supports modeling with different DSLs and is designed to be extensible with additional DSLs. DSLs are based on XML and have to conform to a concrete syntax as specified by an XML-specific grammar (DTD). Models are specified using XML/RDF as instances of a formal ontology and basically consist of first-order logic statements (Villa, 2007, 2001). To each DSL there exists a corresponding executable implementation in the runtime system of IMT, that provides the semantics of the specification which is accessed and executed according to the automated reasoning process. The expressive power of DSLs might range from the mere specification of settings of an pre-programmed model (e.g. setting parameters) to the specification of structure and behavior, e.g. with a modeling paradigm (ODE) or scripting using GPL. Further DSLs exist that allow the specification of experimentation procedures (e.g. optimization), GIS operations as model components and the integration of statistical software for pre- and postprocessing or as model components. It is a distinguishing feature of IMT that an ontology entails information about models that is exploited for automatic generation of

workflows for experimentation, including the resolution of dependencies, transformation of scale and consistency checking.

New DSLs are introduced by description of the semantic structure (formal ontology) and a respective XML-based grammar. Further, the API of the tool must be used specify the translation of model specifications into modules that extend the runtime engine. With the means to define XML-based DSLs, integrate legacy models, web-based resources, GIS, statistical software packages and use some GPL as specification languages for models combined with extensive use of meta-information from ontology, the IMT provides means for integrated modeling that goes beyond the possibilities of other component-based approaches (although the use of some meta-information is state-of-the-art here). GIS functionality is highly integrated, since geo-spatial reference and the treatment of it is built-in (e.g. through automatic scale and reference transformation), thus fully implicit (conceptually tightly coupled). However, DSLs are tightly linked to the implementation of the tool, which requires manual tool-specific implementation of semantics of DSLs, thus reusability across tools is limited. In addition, the concrete syntax of any DSL is XML concrete syntax. Adaptions, e.g. graphical interfaces as concrete syntax of models, have to be implemented manually. According to (Villa et al., 2009), issues that presumably prohibit the adoption of this approach in the EMS community are that only first-order logic is available for modeling, unresolved issues related to bridging different ontologies (paradigms), evolving ontologies and difficulties of modelers with adopting the XML and ontology-based technology.

3.5 Model-driven Engineering and Metamodeling of Computer Languages

This thesis evaluates the application of methods and technology of *Model-driven Engineering* (MDE) to EMS. Model-driven approaches to software and systems engineering can be seen as the natural continuation of developments of programming languages in the sense that these try to deal with complexities of systems and development processes by means of increasing the abstraction from actual machines and systems (Schmidt, 2006). In contrast to code-centric development approaches that mainly are accompanied by higher-level GPL, MDE[17] focuses on relatively abstract models of systems and associated DSLs for modeling (Schmidt, 2006).

> *Model-driven Engineering is an approach to software and systems engineering with "model" and "model transformation" as basic constituting concepts and respective specifications as central artifacts.*

Here, "model transformation" refers to the production of another model - the target model - of the same system and respective model specification at the base of a specification of an original model - the source model.

Three further basic aspects characterize MDE: first, the automated refinement of models and code generation, second, the view-based decomposition and integration of models and, third, the automated model- and language-based assertion of properties of systems under development. These aspects are presented in see the following chapter.

[17] The approach that is referred to as "model-driven engineering" in this thesis is used synonimously to the terms "model-driven development" (MDD) and "model-driven software development" (MDSD) that appear in literature.

3.5.1 Basic Aspects of Model-driven Engineering

Figure 3.3 illustrates the first aspect that is concerned with the automation of the implementation process of complex systems by explicitly representing and relating adequate levels of computation abstraction. Different models represent the system to be implemented at particular levels of abstraction, where more abstract models contain less technical detail than relatively specific models. Models at different levels of abstraction are related by means of automatic *model-to-model transformations* (M2M), where each M2M refines the more abstract model by adding technical detail. Finally, after a theoretically arbitrary number of M2M, a *model-to-text transformation* (M2T, code generation) produces code that conforms to an existing deployable target technology (e.g. GPL, see Figure 3.3 (left)). Original models at the highest level of abstraction are created by modelers using DSLs (i.e. DSL^1). A M2M might transform a model within one DSL (i.e. $M2M_{12}$ transforms from DSL^1 to DSL^1) or the target model might be created with another language (i.e. $M2M_{(n-1)n}$ transforms into language MM_n). The notation *MM* highlights that if a model is the result of a model transformation and human interaction (e.g. lecture, modification) has low priority, the language of the model might be tailored towards computer interpretation, not human-computer communication, with respective concepts and concrete syntax. In this case, the computer language used for model representation is typically not referred to as "DSL", but rather referred to as "metamodel" (MM, i.e. MM_n), since technical considerations dominate the design of the language (see following Chapter 3.5.2 for clarification).

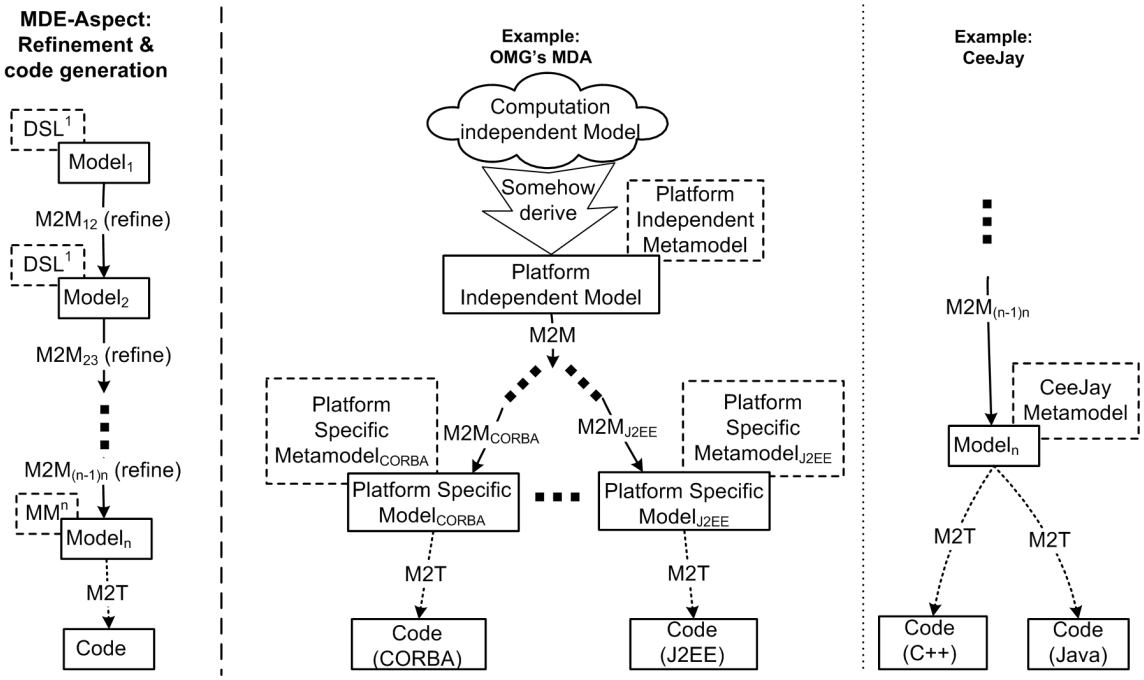

Figure 3.3: The basic aspect of MDE where models reside at different levels of computation abstraction related by refining M2M and M2T(left) and exemplary application patterns (MDA, CeeJay (right)).

In principle, levels of abstraction are arbitrary and specific to applications. However, MDE-based computation abstraction is applied in order to separate complicated basic

technologies (platforms) from higher-level design decisions (models). This should support a separation of concerns, where domain experts define higher-level characteristics of systems and implementation experts their technical realization. Further, through locality of effects, computation abstraction is perceived to reduce maintenance issues that are related to the existence of functionally equivalent and continuously evolving implementation technologies (typically referred to as "middleware platforms", e.g. J2EE, CORBA, .NET), if models capture sensible design patterns that can be reused across a number of implementations and basic technologies (i.e. platforms, GPL). Specific characteristics of lower levels are added by model transformations (M2M, M2T).

Against this background, the influential Model-Driven Architecture (MDA, Miller and Mukerji (2003)) particularly promotes three levels of abstraction for models (Figure 3.3 (middle)):

- *Computer Independent Model* (CIM): possibly informal model of a system with no references to aspects of computation.
- *Platform Independent Model* (PIM): formal computing-oriented model with no conceptual dependencies to particular "middleware platforms" (e.g. CORBA, J2EE). Depending on the development method, the PIM is derived manually, automated or semi-automated from the CIM.
- *Platform Specific Model* (PSM): formal model with conceptual dependencies to a specific platform. The PSM is the result of M2M.

A single PIM is thought to be able to be transformed to a number of different PSMs that are specific to functionally equivalent platforms or different versions of one platform (i.e. for the platforms CORBA and J2EE). Indeed other levels of abstraction are under discussion, such that a common level, with a respective DSL/metamodel, from which it is possible to translate into different object-oriented GPL (e.g. CeeJay-metamodel, Figure 3.3 (right), Piefel (2006)).

Besides the separation of concerns with respect to computation abstraction, a further basic aspect of MDE is a separation of concerns with respect to domains reflecting the heterogeneity of the system to be designed. Different aspects of a system (e.g. physical components of the system, architectural or managerial aspects (concurrency management, distribution of services); Kent (2002)) might be associated with different domains and experts. Respective interrelated view-specific models are specified by means of different DSLs (i.e. Figure 3.4 (left), DSL^A ... DSL^Z). Models are to be combined for further processing.

Holz (2004) identifies two basic strategies for the combination of models: the integration of modeling languages into an integrated language (super-formalism) and the transformation between languages. Depending on the development process, language integration naturally lends itself to parallel use of languages, whereas transformation between languages accommodates serial usage of models and respective languages. A necessary prerequisite for any language integration is the identification of equivalent concepts of languages that are formalized within transformations or common language concepts of the languages to be integrated. Whereas transformation of models requires identification of equivalences of (parts of) target and source languages that does not influence the languages, language integration requires relating several languages, such that the syntax and semantics of the integrated languages might be altered.

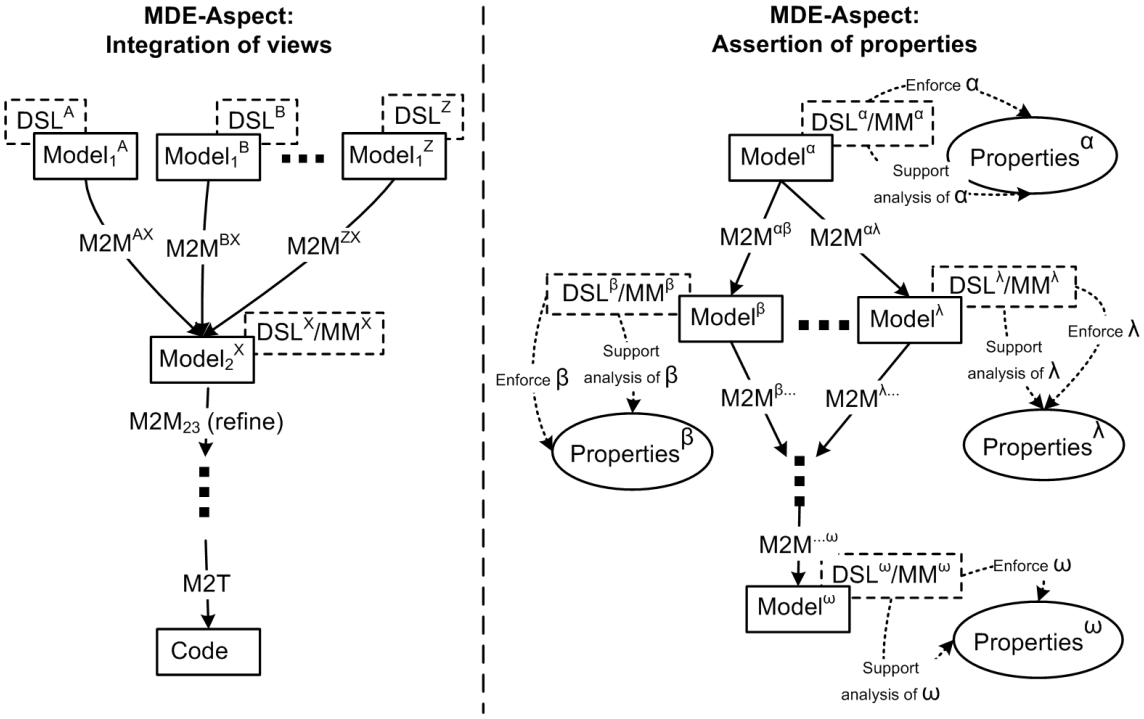

Figure 3.4: The basic aspects of MDE.

a

Figure 3.5 illustrates three variants of language combination based on common concepts. The target language might either incorporate the union of concepts (no duplicates, left), it might incorporate common concepts of source languages and relatively basic concepts that are used to substitute concepts of source languages (middle) or the concepts of one source language are used to substitute the concepts of source languages (right).

The third basic aspect of MDE is related to the development process of systems. Although there is no common agreed-upon model-driven development process, it appears that a fundamental characteristic of any such process would be that several levels of abstraction are used for level-specific analysis of properties (Fondement and Silaghi, 2004; Kent, 2002)). As in existing software engineering processes (e.g. waterfall process, RUP, V-Model) the statement of properties, such as performance properties, precede (or at least parallels) the development of models (Holz, 2004). Major advantages of MDE are expected from the support for automated assertion of a systems properties by means of different kinds of analysis, such as

- automated formal verification,
- automated testing,
- automated model checking,
- validation through experts and
- simulation.

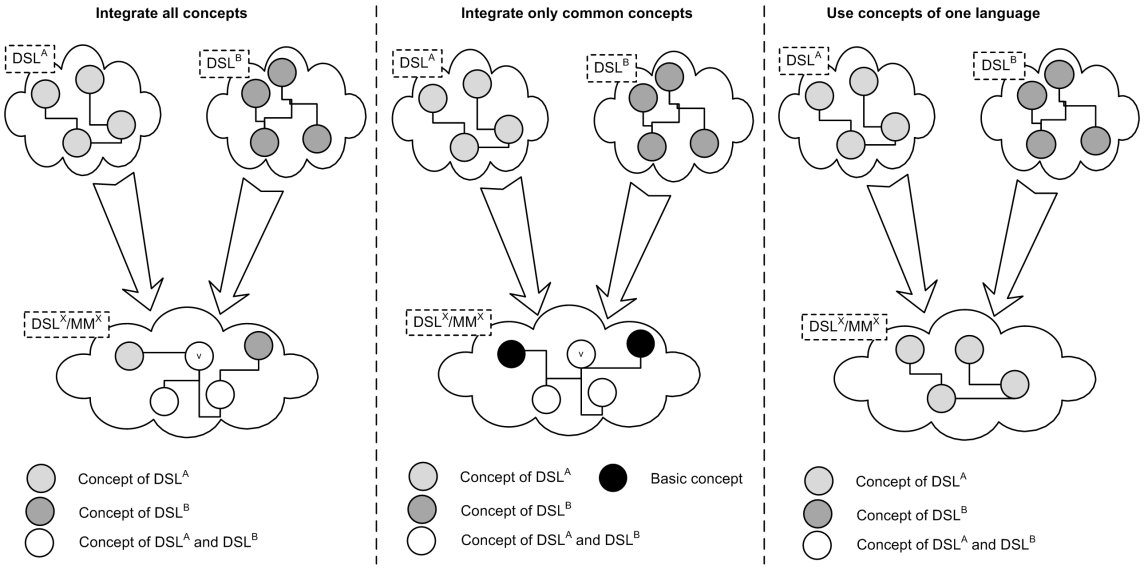

Figure 3.5: Three variants of model integration based on integration of language concepts (based on Holz (2004)).

Such analysis is expected to be realized relatively efficiently when the required information is explicitly encoded in models at an adequate level of abstraction rather than implicitly in code (Figure 3.4, right column). Figure 3.4 (left) illustrates the fact that a number of different models might be necessary to assert a number of properties, with a number different languages (DSLs,metamodels) used, since a specific method of assertion and respective properties might be possible with a specific language only. Further, DSLs can directly enforce properties through syntactical and semantical constraints, such that it is not possible to specify models that violate properties. By this, MDE is expected to support correctness and consistency of (partial) specifications by facilitating "correct-by-construction" rather than "construct-by-correction" (Schmidt, 2006).

As presented, there are different motivations and application patterns (aspects) related to the use models and model transformations as basic constitutional concepts of MDE. Besides the inherent complexity of the MDE approach that indeed directly reflects the complexity of systems and system engineering, a major issue is the efficient realization of this approach since it requires the implementation of language tools for DSLs, which is perceived as being prohibitive using traditional approaches to tool support, such as formal grammars. "Language metamodeling" - in particular object-oriented explicit modeling of abstract syntax - is an approach that is tailored towards the issue of efficient tool support for MDE, as described in the following chapter.

3.5.2 Language Metamodeling

Technically, MDE is based on the possibility to allow modelers and engineers to use adequate modeling languages that are tailored towards high-level system modeling, intermediate representation, code generation and assertion of properties. The efficient realization of supporting tools is a fundamental prerequisite. "Language metamodeling" refers to the explicit modeling of computer languages that particularly aims at efficient, at best automated, implementation of language tools for MDE.

A language metamodel is a model of a modeling language.

A metamodel that is sufficient for fully automatic generation of language tools must encompass models of abstract syntax, concrete syntax, static semantics and dynamic semantics. However, for simplicity this thesis follows the widely used practice to refer to a model of abstract syntax by means of the term "metamodel". Models of static semantics, dynamic semantics and concrete syntax are typically defined at the base of metamodels (abstract syntax) and will further be explicitly referred to as such. In general however, the practice of *metamodeling* encompasses the definition of computer languages by means of metamodels (abstract syntax) and metamodel-based definitions of concrete syntax and semantics as described below.

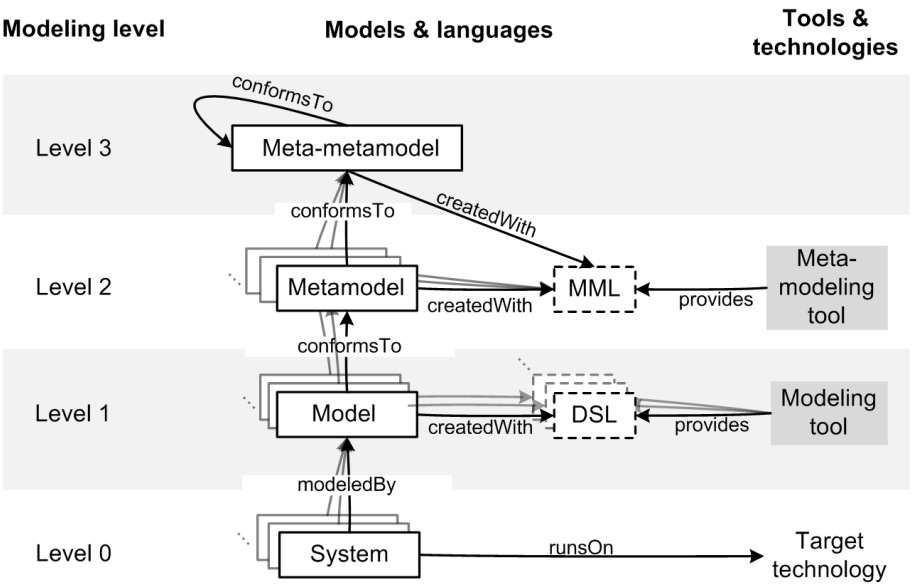

Figure 3.6: Four-level metamodeling.

Figure 3.6 illustrates the typical use of metamodels in a four level architecture. At the lowest level (*Level 0*) reside the actual systems that are to be developed which run on specific target technologies. All models of a particular system reside at *Level 1*, where all models are specified by means of DSLs or generated by means of model transformations. The relation *modeledBy* between *Level 0* and *Level 1*, besides abstraction, denotes the actual instantiation of concepts of models on concrete computing devices and the steps required to implement a running system from a relatively abstract model. The actual realization of *modeledBy* typically is top-down and involves tasks such as compilation of code and its execution, compilation and deployment of software components or direct interpretation. The metamodels at *Level 2* are a models of the languages (*DSL*) that are used to create models at *Level 1*. The meta-metamodel at *Level 3* is a model of the language that is used to create the metamodels of the modeling languages at *Level 2* (metamodeling language, *MML*). The relation *conformsTo* between *Level 1*, *Level 2* and *Level 3* denotes that a model at a lower level must conform to constraints (syntax & semantics) which are specified by the metamodel (meta-metamodel) at the relatively higher level. A defining characteristic is that these constraints are to be enforced and tested through language tools at the respective levels that provide respective languages

(i.e. *Modeling tool* and *DSL* at *Level 1*, *Metamodeling tool* and *MML* at *Level 2*)[18]. *Level 3* is "bootstrapped" meaning that the meta-metamodel conforms to itself and is defined by means of the metamodeling language (MML) it defines.

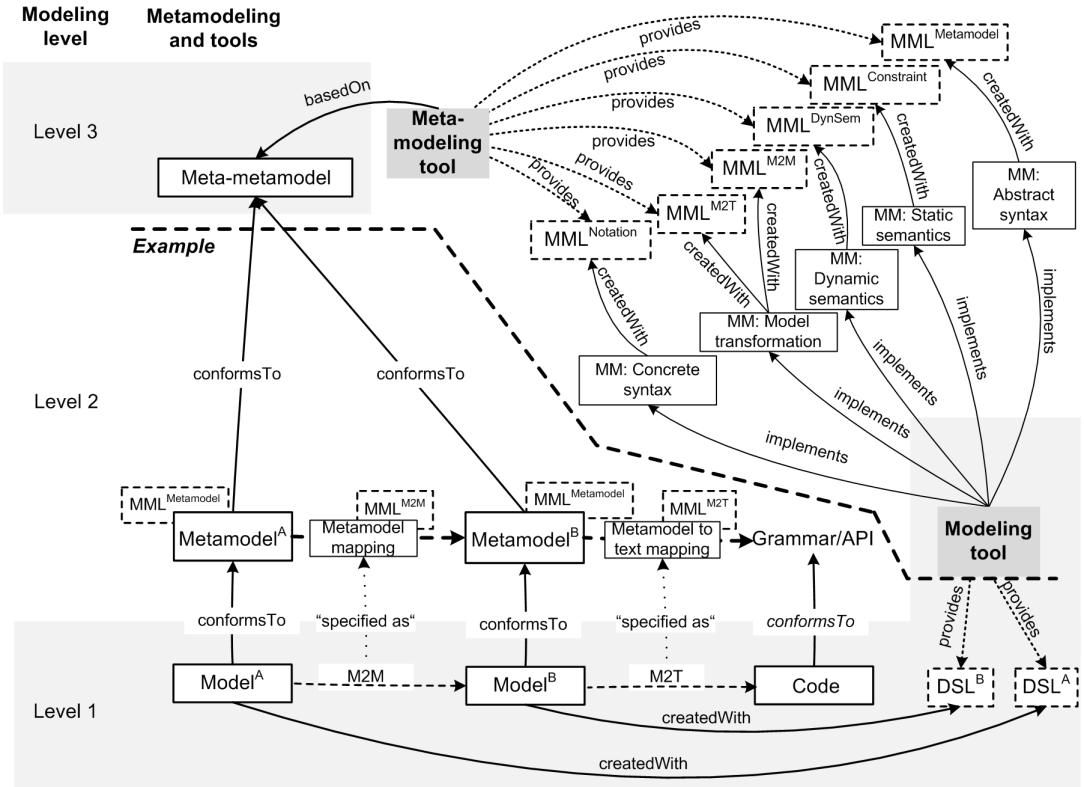

Figure 3.7: Four-level metamodeling.

Theoretically a infinitely great number of meta-levels could be thought of that are related via *conformsTo* relation. In practice however, the four-level approach with two "meta-levels" - levels that are concerned with modeling modeling languages -, a modeling and a system level appear to be sufficient to achieve the goals to use different related DSLs and efficiently implement MDE, thus language tools and transformations[19]. Figure 3.7 further illustrates the practice of the four-level appproach. Based on a meta-metamodel a *metamodeling tool* provides means to manage (create, manipulate, store, read) any conforming language metamodel. Besides the means to specify the metamodel (abstract syntax) using a metamodeling language ($MML^{AbstractSyntax}$) there are means - either special languages or API - to specify all aspects that are needed for realizing MDE (Figure 3.7, upper part):

- Static semantics of DSLs that are not covered by the abstract syntax definition ($MML^{Constraint}$).
- Dynamic semantics of DSLs in terms of interpretation (MML^{DynSem}).

[18]This relation between modeling levels is often referred to as *instanceOf*. For clarity (e.g. instance of in object-orientation) this thesis uses *conformsTo* as proposed in (Bezivin and Kurtev, 2005).

[19]Bezivin and Kurtev (2005) argues that such four-level structure is typical for many other "technical spaces" such as grammar-based language tooling, XML or formal ontologies with RDF.

- Model transformations as mappings between meta-models (M2T) or metamodels and text (M2T) (MML^{M2M} and MML^{M2T}, (Figure 3.7, lower part)).
- Concrete syntax ($MML^{Notation}$).

An important characteristic of implementing the four-level approach is that the provided metamodeling languages (MML) that are directly modeled by the meta-metamodel ($MML^{AbstractSyntax}$) or specific to it (e.g. MML^{M2M}). Modeling tools that implement the respective models of syntax and semantics are automatically generated by the meta-modeling tool. The example in Figure 3.7 (lower part) illustrates that model transformations (i.e. *M2M* and *M2T*) are specified as mappings between metamodels using specific languages (MML^{M2M} and MML^{M2T}). The *modeling tool* provides the DSLs used for modeling and realizes the specified transformations.

There exists a variety of meta-metamodels with different characteristics, however it appears that the incorporation of basic concepts of object-orientation is state-of-the art (see (Kern et al., 2011) for comparison of meta-metamodels). Object-oriented metamodeling is illustrated at the example of the meta-metamodel *Ecore* that has been used for the implementation of DSLs in this thesis.

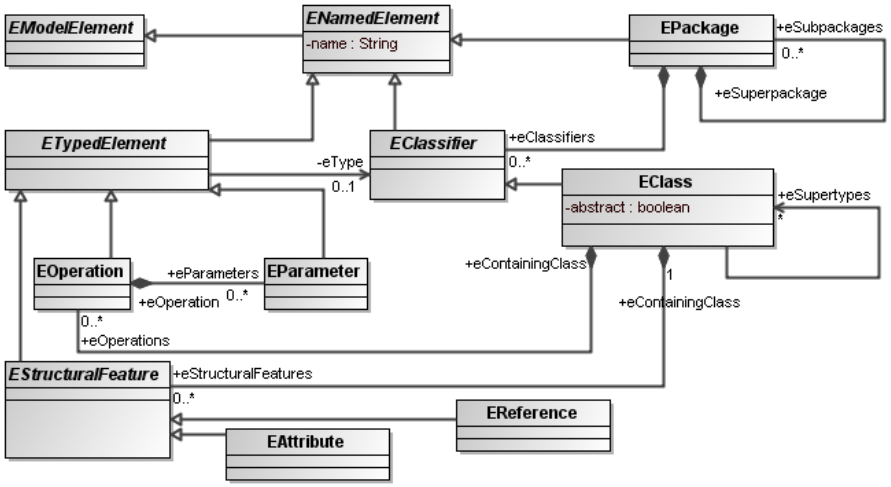

Figure 3.8: The Ecore meta-metamodel in UML class diagram notation (simplified from Eco (2012)) at Level 3.

Figure 3.8 presents the basic part of the Ecore meta-metamodel in UML class diagram notation (see Chapter 2.2.3)[20]. It shows that a conforming metamodel is made up of packages (*EPackage*) that contain classes (*EClass*). Each class may have a number of supertypes (*eSupertypes*) and contains a number of describing features *eStructuralFeatures* that may be typed attributes (*EAttribute*) or typed references to *EClass* (*EReference*). Thus, a conforming metamodel (abstract syntax) is basically defined as an object-oriented class structure as illustrated in an demonstrative metamodel in Figure 3.9 that models a DSL for simple arithmetic expressions.

The model of the abstract syntax of *DSL1* is divided into a number of nested packages (i.e. *DSLCore*, *DSL1*) that contain related classes. Concrete classes specify elements

[20] Ecore is an implementation of the EMOF meta-metamodel that is part the meta-metamodel of MDA with a straightforward mapping of UML class diagram notation to Ecore.

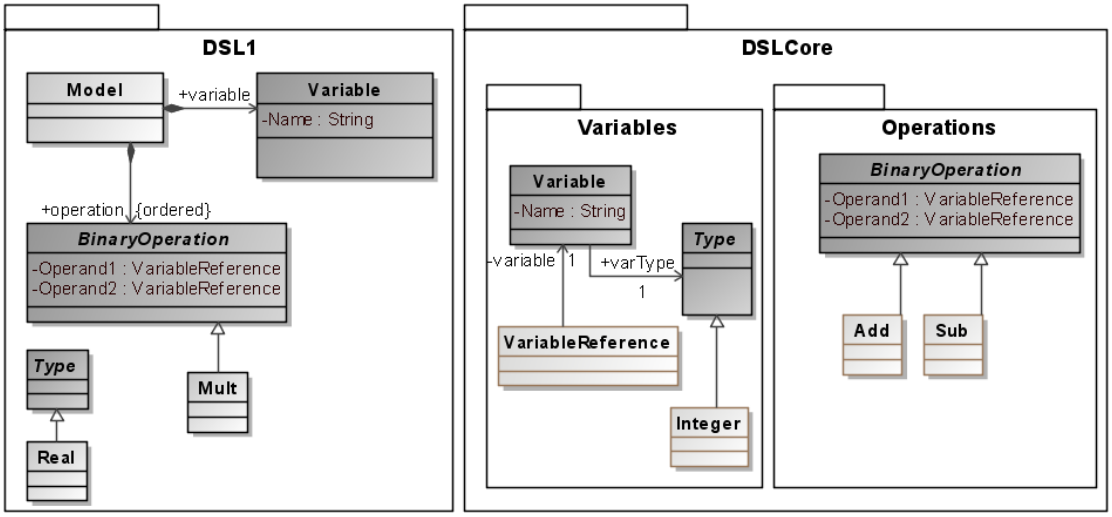

Figure 3.9: Example language metamodel that conforms to Ecore at Level 2 in UML notation.

that may actually appear in a model specification (i.e. *Model*,*Variable*, *Add*), whereas abstract classes structure the conceptualization of the language (i.e. *BinaryOperation*). Package *DSL1* defines the basic structure of model specifications: a *Model* contains the specification of a number of *Variable*s and an ordered set of *BinaryOperation*s - either *Mult*, *Add* or *Sub*- that take two *VariableReference*s as arguments (*Operand1*, *Operand2*). Variables must reference a *Type* - either *Real* or *Integer*.

The example Figure 3.9 illustrates how object-oriented metamodeling supports the compositional definition of metamodels supporting the reuse of partial langauge models. Assuming the package *DSLCore* contains basic concepts that might be useful for a number of DSLs, the Package *DSL1* only specifies concepts specific to *DSL1* and uses concepts from *DSLCore* (dark gray) by either referencing (i.e. *+variable*) or subclassing (i.e. *Real*). Given common concepts, this technique is straightforward to realize, in particular when DSLs and language tools are implemented from scratch. In general, object-oriented metamodeling supports straightforward means to formalize the combination of languages, based on common concepts as illustrated in Figure 3.5. By this, object-oriented metamodeling is perceived to support the reuse of existing specifications of abstract syntax and possibly concrete syntax, transformations and semantics for the purpose of reduction of efficiency and reuse of well known patterns (Emerson and Sztipanovits, 2006). Further, the orientation towards abstract syntax - in contrast to concrete syntax - is perceived to cause less effort with respect to tool implementation, lower the degree of specificity of DSLs to tools, since it supports the derivation of different language tools for the same metamodels and respective models, meta-modeling tools support the same set of metamodeling languages (in particular compared to grammars, see Chapter 3.1). Further, the language definition based on abstract syntax is perceived to support adaption and should enable the conceptualization and formalization of language combination patterns and techniques (Holz, 2004).

Emerson and Sztipanovits (2006) illustrates three common techniques for the composition of existing object-oriented metamodels:

- *Metamodel Merge*: Metamodel Merge is based on the assumption that conceptual spaces of languages intersect. Common elements of metamodels are algorithmically identified based on similarities and replaced by one respective new element.
- *Metamodel Interfacing*: Metamodel Interfacing is based on the assumption that conceptual spaces of languages are related. New metamodel elements are created that relate existing elements of the combined languages.
- *Class Refinement*: One DSL prescribes the coarse structure of model, another DSL is used to specify refinements of the coarse structure. The metamodels of the languages are related such that the elements of one metamodel are contained in one element of the other language.

Whereas the development of new languages can use the well known features of object-orientation and realize its benefits, the combination of existing DSLs must deal with the issue that existing languages might obtain altered and extended concepts spaces and notations with the need to adapt existing tools, in particular when common conceptual core is realized by extension of existing languages or conceptual intersection is defined (see Holz (2004) for elaboration). However, with the object-oriented structuring of language metamodels (abstract syntax), the definition of other language aspects might follow the decompositional structure of the abstract syntax.

Listing 3.1 illustrates the use of the *Check* language (see Efftinge et al. (2007)) that can be used to specify constraints ($\text{MML}^{\text{Constraint}}$).

Listing 3.1: Constraint definition for Ecore metamodels with the *Check* language

```
1 context DSL1::Model ERROR "Wrong number of variables." :
2   this.variable.size >= 2 && this.variable.size < 10
3
4 context BinaryOperation ERROR "Operands must be of same type." :
5   this.Operand1.variable.varType.metaType !=
6     this.Operand2.variable.varType.metaType;
```

Whereas lines (1/2) formalize the constraint that a model must contain between 2 and 9 variables, lines 4-6 show a rule that enforces that operands of a *BinaryOperation* must have the same type. Please note how references to the metamodel of *DSL1* (i.e. DSL1:Model, variable) are made based on language constructs that exploit the object-oriented structure enforced through the meta-meta-model (i.e. *context* for referencing classes, *::* for addressing *EPackage*, *.* for addressing *EAttribute* and *EReference*). Further languages used in this thesis are languages *Xtend* for model transformation and *Xpand* for M2T transformation. These languages use same metamodel navigation mechanisms and allow a structuring of specifications along the package-structure of a metamodel.

Listing 3.2 shows a partial metamodel-based definition of a textual concrete syntax for *DSL1*. The language used (*TEF*, Scheidgen (2009)) integrates BNF-like formal grammar rules with references to Ecore metamodels, where grammar rules are referenced at left side of ":" and the metamodel at the right hand side. Navigation of the metamodel is realized by language constructs based on the meta-metamodel (i.e. *composite* for *EAttributes*, *element* for *EClass*, see Scheidgen (2009)).

Listing 3.2: Concrete syntax definition for Ecore metamodels with the *TEF* language

```
1 Model:element(Model) -> "Model"
2   (VariableDeclaration:composite(variable)";")*
```

```
3    (BinaryOperation:composite(operation)";")*;
4
5  VariableDeclaration:element(Variable) ->
6    "Variable" TypeByRef:reference(varType) IDENTIFIER:composite(name)";"
7
8  BinaryOperation -> Add;
9  BinaryOperation -> Sub;
10
11 Add:element(Add) -> VarReference:composite(Operand1) "+"
12       VarReference:composite(Operand2)
13 Sub:element(Sub) -> VarReference:composite(Operand1) "-"
14       VarReference:composite(Operand2)
```

Listing 3.3 presents a possible model in concrete syntax as defined by Listing 3.2 that conforms to the language metamodel. A *Model* is specified that contains the variables *Var1* and *Var2* of *Type Integer*. The *BinaryOperation Add* is applied.

Listing 3.3: Example model for the exemplary Ecore-based language *DSL1*

```
1 Model
2    Variable Integer Var1;
3    Variable Integer Var2;
4    Var1 + Var2;
5 ;
```

The examples illustrate how object-oriented meta-modeling aims at supporting modeling with several DSLs by facilitating efficient implementation of single and various combined DSLs through automation and reuse and combination based on concepts, rather than notation. However, there is a variety of available meta-metamodels and respective meta-modeling tools (Kern et al. (2011); Pfeiffer and Pichler (2008); Tolvanen (1998) provide overviews and comparisons). Integration of DSLs across meta-metamodels is theoretically possible if meta-metamodels share similarities, but this requires translation of metamodels and models (some translations between different meta-metamodels have been defined so far, but this field is subject to current research, see Kern et al. (2011)). Although literature suggests that the assumption appears to be justified that there is some agreement with respect to most basic object-oriented properties of metamodels (abstract syntax), there appears to be a variety with respect to heterogeneity of object-oriented MML ((Kern et al., 2011)), in particular MML for other language aspects[21]. Further, meta-modeling tools vary significantly with respect to supporting functionality. Meta-modeling tools range from simple text editors for the specification of syntax, semantics and transformations to editors with syntax highlighting, code completion and validation (Pfeiffer and Pichler, 2008). The heterogeneity of MML and tools impedes the reuse of metamodels across different metamodeling tools. Although the development of MML aims at generality (expressive power) and single tools may support a variety of MML (e.g. EMF, EMF (2012a)), metamodel composition, translation and finally reuse of partial metamodels is subject to current research and ultimately subject to agreement (e.g. through standards). Further, the metamodel-based provision of some important functionalities of language tools that are commonly provided by modern IDEs for GPL, such as debugging and profiling, are

[21] The example of Ecore and Ecore-based MML illustrates heterogeneity. $MML^{Constraint}$: Check (Efftinge et al., 2007), OCL (EMF, 2012b); MML^{M2M}: QVT Operational (Dvorak, 2008), QVT Declarative (QVT, 2012), ATL (The Eclipse Foundation, 2012), Xtend (Efftinge et al., 2007); $MML^{Notation}$: TEF, XText (xText, 2012), GMF (GMF, 2012)); MML^{M2T}: Xpand (Efftinge et al., 2007), JET (Popma, 2012).

not developed to a degree such that they can be used with common, non-research, metamodeling tools.

3.5.3 Design of DSLs

The primary design goal of MDE and thus, the application of DSLs, is to facilitate gains in productivity at the production of systems through the automation of the implementation of systems and modeling tools and further the reuse of system models and language definitions in combination with explicit representation of adequate levels of abstraction. The technical aspects of metamodel-based automation and and reuse have been presented (Chapter 3.5.2). Other aspects related to the design of appropriate DSLs are perceived as relatively poorly developed (e.g. Gabriel et al. (2010); Wu et al. (2010); Karsai et al. (2009)). Generally, design criteria stated for GPL apply to DSLs (e.g. regularity, adequate tool support etc., see Chapter 3.2). However, it is argued (e.g. Mernik et al. (2005); Karsai et al. (2009)) that the technological changes (e.g. availability of memory, computing power, language tools) and the domain-specific educational background of intended users of DSLs allows and requires a shift of focus, away from primacy of efficiency considerations with respect to compilation and execution towards efficiency with respect to modeling.

The discussion of non-technical design characteristics of DSLs is framed into "guidelines" for development of DSLs (e.g. Mernik et al. (2005); Wile (2003)), methods for quantitative (e.g. Wu et al. (2010); Sprinkle (2010)) or qualitative evaluation (e.g. Haugen and Mohagheghi (2007)) of DSLs and further, the combination of formal ontologies and DSLs.

In general, quantitative metrics depend on a formal statement of quantifiable characteristics of models that are to be optimized by the design of a DSL. The basic strategy to optimize productivity is the minimization of effort. Whereas (Hill, 2011) defines "modeling effort" as the "the number of steps it takes a modeler to complete a highlevel modeling goal (or a task)", Wu et al. (2010) suggests a relatively comprehensive measure of effort with a number of different factors: the number of concepts (e.g. nodes, edges, attributes, auxiliary variables, statements, methods etc.) and possible control flows of a model. Further, there are empirically defined cognitive costs of single language constructs which are aggregated at the base of models. Finally, factors such as the number of keystrokes, clicks, drag-and-drops, cpu and memory usage are considered (Wu et al., 2010). Although quantitative metrics provide background information when (different versions of) DSLs are to be compared, they do not provide characteristics of DSLs by itself: the number of concepts used and possible control flows, further the number of necessary atomic interactions (e.g. keystrokes) is a function of the generality of a language, which depends on the variety of problems to be addressed by a language, thus minimization is only valid when the relevant solution space remains unaffected by it. Without formal statement of requirements and design goals optimization is subjective. Although metrics might reflect improvements of DSLs, they provide no means to explain them.

In contrast to metrics, guidelines typically put the development process of DSLs at the center of considerations about the quality of DSLs. However, based on experience, Karsai et al. (2009) states that " [...] many design guidelines cannot be translated in automatic measures [...]". Publications provide sets of guidelines with varying specificity (e.g. Mernik et al. (2005); Wile (2003); Karsai et al. (2009)). Since the technical requirements for DSLs are well accepted - and relatively well-described - guidelines typically give suggestions in form of design goals. Essentially, guidelines suggest

- the incorporation of domain-specific concepts, existing concepts and notations that

must be identified with domain experts,

- the avoidance of "improvements" and extensions of domain concepts and overgeneralization as a typical characteristic of GPL,
- the alignment of the structure of DSLs with organizational structures and roles, e.g. by means of adequately combined DSLs and
- the alignment of DSLs with the educational - possibly non-technical - background and goals of users, that takes into account learning curves and expectations towards technologies.

Thus, guidelines do not provide concrete and fundamental characteristics of DSLs with respect to domain-appropriateness and concrete methods to achieve adherence are not given[22]. This might be explained by the experience that the characteristics of DSLs are specific to domains, users, goals and that further there is seldom agreement on characteristics of "good" DSLs among users (Karsai et al., 2009)[23]. Against this background, Karsai et al. (2009) proposes that guidelines have to be discussed and balanced in the context of specific domains and DSLs. Thus, essential guidelines are abstract in that they abstract from underlying concrete design decisions (such as the level of generality, the knowledge and goals of users etc.), where the choice of which typically requires case-specific balancing.

Recent works elaborate on the relation of formal ontologies and DSLs since they are perceived to share a common conceptual background and provide methodological improvements with respect to the alignment of DSLs and domains:

> *According to Gasevic et al. (2006) a formal ontology may be defined as a formal, explicit specification of a shared conceptualization that includes a vocabulary, semantic interconnections and rules of inference for automated reasoning.*

Thus like DSLs, its a defining feature of formal ontologies that they formally capture domain specific knowledge. However, Gasevic et al. (2006) argues that the methods for knowledge elicitation and formal representation are relatively well developed in the field of ontologies, such that integration with DSLs promises to be beneficial with respect to domain-appropriateness. Formal ontologies are associated with languages for the specification of ontologies in a way that automated reasoning can be applied and methods and languages for the elicitation and formalization of knowledge (ontology development) with specific supporting tools (Gasevic et al., 2006). There are two main aspects to this integration. First, the application of methods and supporting tools for knowledge elicitation and the technical integration of ontologies and DSLs. A number of methods for knowledge engineering and domain analysis (e.g. DSSA, FODA, ODM) support the explication and formalization of knowledge from heterogenuous sources that eventually leads to the specification of a formal ontology (Ceh et al., 2011). Although there are numerous languages and associated reasoning (inference) algorithms (of which some are Turing

[22] Although guidelines typically encompass concrete guidelines, such as to "allow comments" and to "ask experts", these do not affect the conceptual core of DSLs.

[23] For example, a common notion associated with DSLs is that of a DSL being "small", with a small number of concepts (Haugen and Mohagheghi, 2007). However, a "small" language might be achieved by means of a high degree of generality and orthogonality. Given a problem domain with a great number of specific concepts, a "small" DSL necessarily introduces generality that introduces implicitness in model representation, thus conflicts with the primary purpose of DSLs.

powerful), ontologies are typically based on logics designed for the purpose of inference of conclusions from facts such as logical consistency, logical implication, subsumption, equivalence (Parreiras et al., 2007).

Formal ontologies and associated tools for formal and semi-formal methods of domain analysis might assist the early development DSLs since they aim at identification of terminology, commonalities and variabilities of domains (Tairas et al., 2009)[24]. A formal ontology can be used to automatically derive language syntax (Tairas et al., 2009; Ceh et al., 2011). Ontologies can be used to state similarities (domain appropriatness) between DSLs and a domain (Guizzardi et al., 2005). Laarman and Kurtev (2010) propose a meta-metamodel that allows to specify a domain ontology along the abstract syntax within a metamodel. With using a respective language, model elements are not only elements according to abstract syntax, but also to domain ontology, which can be automatically checked by modeling tool. Walter et al. (2009) propose to describe DSLs by means of formal ontologies so that reasoning capabilities of ontologies can be used to check the consistency and constraints of models, verification and debugging of models and interactive provision of suggestions to users of DSLs[25].

However, (Ceh et al., 2011) states that "even if the domain analysis is done with a formal methodology, there are not any clear guidelines on how the output from domain analysis can be used in a language design process (Ceh et al., 2011)." Gasevic et al. (2006) concludes that "even with the most advanced ontology development languages, environments, and methodologies, a major problem in ontological engineering still remains in the area of knowledge acquisition and maintenance - the collection of concepts and relations in a domain, achieving consensus on them among the domain experts and other interested parties, and frequent updates due to the dynamics of the knowledge structure of the domain and its unpredictable changes over time Gasevic et al. (2006)." Thus, although it appears that the goal of capturing domain specific knowledge appears to be a central issue and methods and tools exists for its acquisition, besides technical considerations, concrete qualities of DSLs and metamodels are subject to subjective application-specific balancing of factors by language engineers.

3.6 Conclusions

Chapters 3.3 and 3.4 show that there is some tradition of applying innovations from the field of software engineering to EMS. Although methods and tools from software engineering suggest some generality, the number of distinct developments in the field of EMS indicates that there exist significant differences. However, there is no characterization of the commonalities and differences between the domains with respect to the application of methods of software engineering to EMS, in particular so that characteristics of the

[24] Ontological modeling, although it might not be refered to as such, is typically part of Software Engineering methods in early stages (such as OOA) as it involves characterization of main concepts of an application domain (Gasevic et al., 2006). In particular object-oriented modeling with class structures exhibits formal representational similarities, which however rather focuses on modeling behavior than ontological structures (Tairas et al., 2009).

[25] The technical integration of formal ontologies and language metamodels is typically based on an integration based on exploitation of a four-layer structure of MDE and ontology languages and technologies, such that ontologies can be related to language metamodels based on relationships between meta-metamodels of MDE technical space and ontology technical space (Gasevic et al., 2006) or integration of ontological modeling in one meta-metamodel (Laarman and Kurtev, 2010).

application of MDE in EMS can be derived from it. The theoretical background of system-theoretic M&S presented in Chapter 2 and characteristics of existing tools presented in Chapter 3 however suggest such commonalities and differences (see Chapter 3.6.1 below). Further, it is argued that the general aim to raise the level of abstraction of the description of models requires the identification of the specific role of MDE beyond purely technical considerations of feasibility (see Chapter 3.6.2 below) and derive specific suggestions for the design of DSLs in EMS (see Chapter 3.5.3 below).

3.6.1 M&S in Engineering and EMS: Basic Commons and Differences

Modern Software Engineering (and M&S) and EMS are well aligned in that both widely adopt the theoretical foundations of GST and Dynamical Systems. Thus, well-accepted theoretical, methodological and technical approaches to deal with general issues find widespread application in both domains (experimentation procedures, universal simulation algorithms, components, object-orientation etc.). Basic issues are generally related to the structural complexity of systems and their models, the requirement of implementing simulators correctly and efficiently and the need for collaborative modeling. However, the example of MDE and recent developments in EMS show that both domains also widely share unresolved issues that are related to the technological complexity, heterogeneity and evolution of currently available basic technologies. These technologies typically come along limitations with respect to reusability, implementation (modeling) efficiency and transparency. The approaches "MDE" in software engineering and "Integrated Declarative Modeling" in EMS both explicitly address these issues by promoting a separation of concerns with respect to the dimensions computation abstraction and model decomposition along a system's perceived structure, while explicitly promoting the incorporation of existing basic technologies. Thus, there is a general considerable overlap with respect to motivation and approach in software engineering and EMS.

At a more detailed view, EMS exhibits distinguishing features. Apparent differences are the degree to which geospatial reference is considered and the educational background of modelers, which is typically non-technical in EMS. A rather fundamental difference is that software and systems engineering appears to be relatively concerned with systems analysis and systems design, thus the analysis of unknown behavioral characteristics which are derived from known or planned structural and behavioral characteristics of a system. Methodologically software engineering is largely concerned with the acquisition and assertion of predefined functional and non-functional properties of technical systems under development, where simulation is one of several methods of analysis. Software engineering processes suggest a systematic evaluation of properties of systems, which motivates the parallel use of different models using different paradigms that allow different kinds of analysis for assertion of different properties, where models are related by means of possibly automated transformation of the inner structure of models. In contrast, EMS puts relatively great emphasis on systems inference with a relatively great degree of epistemic uncertainty (e.g. lack of data and structural knowledge) at lower levels of the model lifecycle (I & II, see Chapter 2.2.4). The gathered knowledge lays the theoretical and practical basis for systems analysis and system design at higher levels of the lifecycle (III & IV). However, the use of models at levels III and IV (environmental management) is largely concerned with scenario analysis and communication processes among stakeholders with non-technical and non-scientific educational background.

Transparency is a fundamental requirement of model specifications throughout the life-

cycle, since it determines the appropriateness of models for creative model-based reasoning and sets the framework for evaluating the credibility, thus appropriateness of models in particular against the background of reuse. Thus, the importance of transparency is rather fundamental and reaches beyond considerations of efficiency and ease of use. Although different models or submodels of a system in EMS at different scales with different paradigms might exist in particular for management purposes, their relationship is rather that of substitution - thus they represent alternatives, not complements - and each model with according modeling paradigm and level of abstraction is subject to experimentation and empirical evaluation on its own, rather than the transformation of the inner structure of models[26].

3.6.2 MDE-based Tools for EMS

Simulation technology for EMS facilitates the execution of experiments and experiment series that follow an experimental procedure that basically encompass the tasks of data pre-processing followed by the execution of a simulator and the post-processing of data, where the different tasks are connected via the conceptually loosely coupled exchange of data. State-of-the-art M&S software is typically composed of software units that have been developed with a specific theoretical background within a specific domain (e.g. simulation, GIS, statistics, Figure 3.10, middle), further referred to as "implementation domains" that are formed by the intersection of relatively universal mathematical methods and their computational realization, as opposed to "domains" with respect to system modeling that are formed along the perceived structure of systems, related methods and knowledge (e.g. hydrology, meteorology, ecology).

Whereas state-of-the-art component-based approaches suggest a loose coupling of software units between and within the computation tasks of experiments (e.g. scientific workflows), the need for tight coupling is evident in the case of software units that are implementations of model components[27]. Where loose coupling can be applied, the component-based approach naturally aligns software domains with the computational tasks of experimentation and the conceptual decomposition of the system and experiment. However, considerations of implementation and execution efficiency might require that models of Dynamical Systems are implemented by means of software units which are themselves composed of tightly coupled software units (e.g. extensible GPL and shared memory) of different implementation domains, such as simulation and GIS in the case of spatially explicit Dynamical Systems. In this case, loose coupling may be prohibited by considerations of computing efficiency and technical complexity of its implementation. Thus, although component-based approaches align with decomposition according to reusable software units with respect to computational tasks of experimentation and model composition, they do in general not align with implementation domains since the implementation of components requires tight coupling of implementation domains (Figure 3.10, middle). With today's tools for integrated EMS, the choice of the type of integration and its technical realization is the responsibility of modelers.

[26] The kind of modeling where a behavior of a generative model is itself modeled by means of a descriptive (e.g. statistical) model that can be used as a generative model with less accuracy, is referred to as *metamodeling* (see Villa-Vialaneix et al. (2012)). Please note that this is not related to language metamodeling.

[27] Tight coupling might be necessary in pre- and postprocessing, the detailed evaluation of these however is beyond the scope of this thesis.

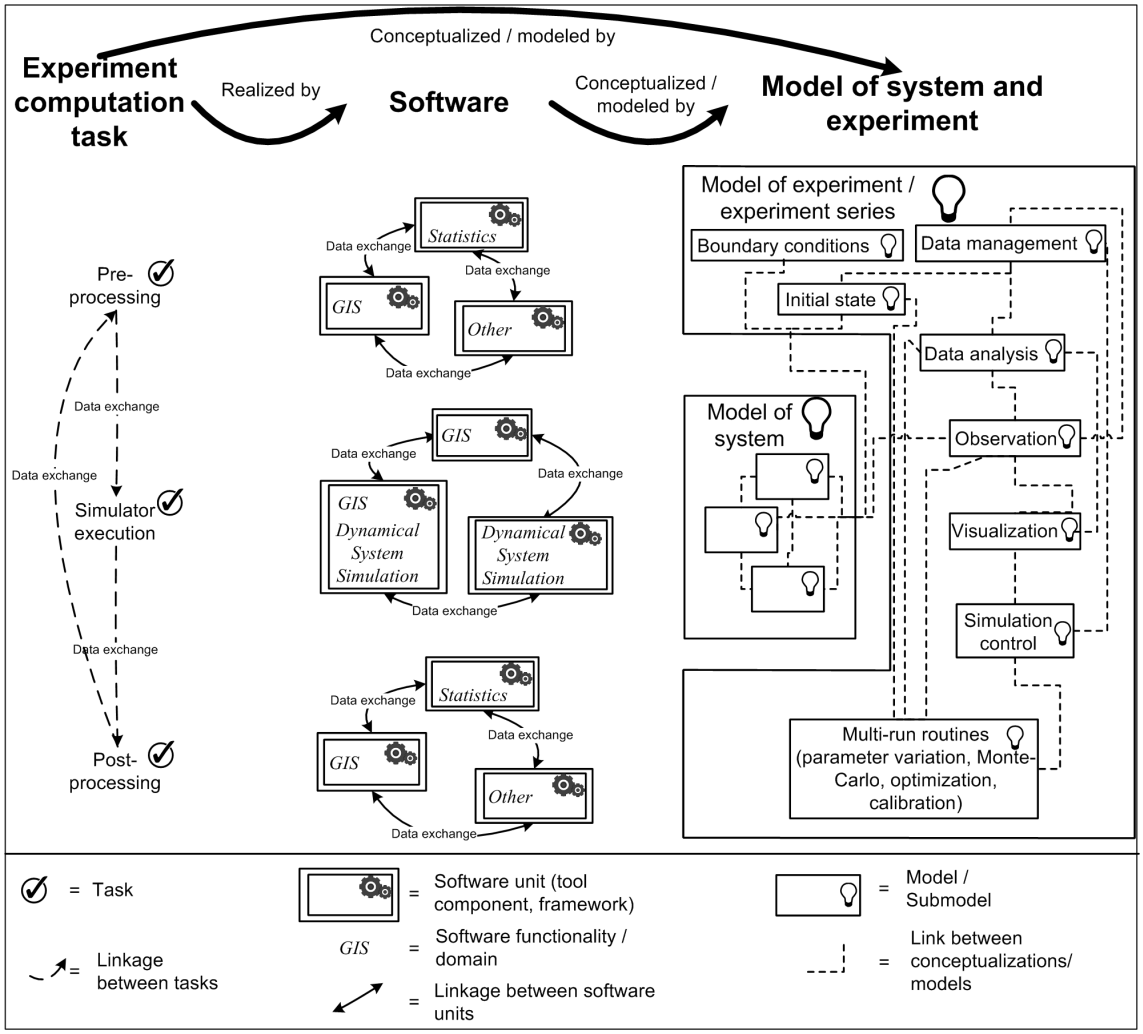

Figure 3.10: GIS and simulation functionality for experimentation.

At a higher level of abstraction, the theory of M&S and the characteristics of existing tools suggest that for experimentation there exist two separate, though connected models: the model of the system under study and a model of the experiment (Figure 3.10, right). In practice, these models might be explicitly or implicitly represented depending on the form of specification. Raising the level of abstraction from programming to modeling requires the transition of GPL with internal DSLs (Figure 3.10, middle) to the explication of models and relations between models using DSLs (Figure 3.10, right). Whereas the model of the real system is a Dynamical System that is conceptualized with a respective modeling paradigm, the model of the experiment specifies aspects of experimentation, such as: multi-run routines, observations, data analysis, visualization and the specification of initial states. Experimentation might further encompass technical aspects of data management (e.g. access) and simulation control (e.g. parallel execution of experiments). These aspects constitute submodels of a model of an experiment (series) which are indeed related, since for example data analysis is based on observations, and both need to be handled based on data management, which is the base of automated multi-run experiments that may require simulation control and the automated setting of initial states. With the rise of the level

of abstraction suggested by MDE and Integrated Declarative Modeling, the alignment of conceptualizations of system models experiments with software domains and different technologies might be internalized in respective modeling tools. Further, the approaches can be understood to suggest the explicit representation of different models using DSLs.

Although extensible GPL enable tight coupling of different software domains by means of internal DSLs (e.g. through object-orientation) that provide some degree of abstraction and domain orientation, this approach is perceived to have shortcomings with respect to transparency and reusability. Component-based approaches address the issue of reusability, the issue transparency however remains, in particular when tight coupling is to be used within single software units. External DSLs address the issue of explicitness and transparency, but up to today the issues of efficiency of tool implementation and reusability of models appear to be prohibitive. Approaches based on language meta-modeling however address this issue by providing means to provide language definitions that promise to reduce these costs by means of automation of tool implementation and reuse of parts of language definitions (syntax and semantics).

Although a number of DSLs has been developed yet, the design of DLSs, thus the characteristics of concepts they embrace, are unclear, besides fundamental technical considerations and guidelines that require application-specific balancing. In the following the epistemic-cognitive background of scientific model-based reasoning and MDE is used to identify more specific characteristics and the relation to MDE.

3.6.3 MDE, Type Hierarchies and Transparency

The theory of model-based reasoning provides an epistemic-cognitive background, against which properties of the alignment of DSLs, MDE and EMS can be discussed.

M&S, MDE and Type Hierarchies

There are a number of well-established notions in system-theoretic M&S that may be perceived as types or their representations. First, a model from which a simulator is derived is a type of which its simulator is a materialization[28]. Models prescribe first-order properties that are to be realized by its materialization, while there are no second-order properties. Meta-properties, such as natural laws or structural properties might be explicitly represented (e.g. as constraining equations) or implicitly, when the structure of a model leads to adherence of a meta-property (e.g. model equations and state are such that observed behavior follows a natural law,which is not directly represented). Further, the perceived structure of a model, thus the concepts and relationships as representants of properties of real systems, is a meta-property. Second, a parameterizable model is a type in that it prescribes concrete structural and behavioral properties (e.g. a set of equations) as a meta-property, first-order properties (e.g. parameters with preset values) and second-order properties by means of variable and parameters that may be chosen from a range of values (initial state). Meta-properties (explicit or implicit) have to apply to to all subtypes and respective possible models. Third, the modeling language used for the specification of a (parameterizable) model is a representation of a type, since it pre-

[28] Please note that in the context of type hierarchies the term "model" refers to materialization of a type, whereas in this thesis the term "model" refers to an abstract entity that is represented by means of materializations (e.g. specifications, simulator).

scribes a set of properties that hold for all models that can be specified by means of it[29], which cannot be formalized. The provision of first-order properties may be asserted by constraints, second-order properties are the degrees of freedom offered by the language. These degrees of freedom encompass the possibility to set values of required attributes, but also the specification of required complex model elements, such as equations or functions. Typical explicit meta-properties prescribed by modeling languages are structural properties that follow the language's modeling paradigm. Although, some paradigms are particularly associated with specific properties (e.g. System Dynamics and non-linearity), thus explicit meta-properties are related to implicit meta-properties, it is typically the nature of scientific studies to discover these relationships, so that the class of models denoted by a paradigm encompasses models to which implicit meta-properties apply and those to which they do not. Such meta properties are typically not stated explicitly and it is subject to the introduction of subtypes to explicate these relationships, if present. However, implicit meta-properties of types represented by DSLs might in general be explicitly stated by means of first-order properties for the purpose of documentation (e.g. in propositional form).

Leaving cognitive aspects aside, any abstraction with a model as subtype may be perceived as a type. However, what distinguishes types from mere abstractions is their reasonable use within model-based reasoning processes that is basically characterized by the need to interlock technical necessities with cognitive processes. Two main aspects can be distinguished: first, the properties of (the representation of) types, and second, the relations between types. Some relations between types subject explicit consideration in M&S with associated methods and forms of representation. With the notion of "correctness" there is a well-established relation of model and simulator that is to be established by means of verification. Whereas the use of existing technologies provides some "correct-by-construction" verification for the relation between model specification and simulator, the relation between model and model specification is widely subject to the explicitness of model representation. The parameterization of parameterizable models is subject to explicit consideration in scientific discourse and there exist accepted scientific and experimental procedures that explicitly deal with the set-up of parameterizable models (e.g. parameter values, data). In some cases, these procedures may be algorithmic (e.g. optimization), however the transparency of experiments, models and data, thus explicitness of representation is a prerequisite. The relation of paradigms and models is generally such that the paradigm constrains modeling to a specific forestructure, the appropriateness of which is, besides basic technical considerations of efficiency and universality, mainly subject to cognitive aspects of model-based reasoning, thus explicitness of representation. The notion of transparency can be cast into terms of type hierarchies.

Model-based reasoning and MDE

Type hierarchies are perceived as the medium of model-based reasoning processes since they represent the basic cognitive structure for reasoning. Types denote the basic entities of reasoning which are related via abstraction relations as a result of different forms of reasoning. Since these types are direct subject to operations of reasoning, it is assumed

[29] The idea to relate ontological type hierarchies and DSLs is not new as shown by the developments that relate formal ontologies and DSLs. However, formal ontologies refer to a specific subset of relationships tailored towards automated reasoning. With this respect type hierarchies are perceived to be more general, since they encompass any operations related to model-based reasoning (see Chapter 2.4.4).

that the more representations of types match respective types, which might be cast in in terms of transformation costs, the greater is the explicitness of representation, thus adequacy of representations (e.g. DSLs), given the technical requirements are fulfilled. If so, concrete forms of representation of types and their relationship have direct influence on adequacy of the design of DSLs. If DSLs are perceived as (representations of) nodes in a hierarchy, the properties of concepts of DSLs are not to be evaluated in isolation, but also how DSLs relate to other nodes (e.g. models) in a hierarchy.

MDE with object-oriented meta-modeling basically provides two specific ways to represent abstraction that can be aligned with model-based reasoning[30]: first, inheritance or composition relations between metamodel elements (classes) within a metamodel and second, model transformations between metamodels (M2M, M2T). Generally, model transformation naturally aligns with computation abstraction, where the meaning of a paradigm is given in terms of another possibly more general paradigm. The hierarchical ordering of object-oriented language metamodels naturally lends itself to representation of formal inheritance relationships of type hierarchies by directly representing first-order, second-order and explicit meta-properties by means of object-oriented class hierarchy: In general, a class might represent a modeling paradigm as a conceptualization of a type. A concrete class might represent a DSL of which abstract superclass(es) might represent relatively general paradigm(s). By this, metamodels allow explicit representation and implementation of *paradigm specialization*. In contrast to first-order and second-order properties, multiple inheritance of meta-properties can in general not be automated by means of inheritance rules, since it typically requires combination by means of creative reasoning (e.g. combination of implicit meta-properties or thought patterns of paradigms). However, inheritance lends itself to paradigm combination where the combined DSL is a combination of different paradigm-specific metamodels using existing combination techniques (e.g. common metamodel elements, see Chapters 3.5.1 and 3.5.2).

Practically, given a new type is discovered as a subset of a type (represented by a DSL), it is possible to a introduce a DSL representing this new specific type that formalizes and enforces its specific properties. If the set of language concepts remains unaffected, one way to implement this is to specify a new set of more restrictive constraints on the existing metamodel that only allows conforming instances, the respective meta-property might be documented as a first-order property of the subclass. Second, the set of language concepts is altered, by means of the introduction of a new type that is related to the original type by means of direct subclassing, if additional language concepts are introduced or by the introduction of common supertype and derivation from it (when language concepts are removed or substituted). The semantics of the computationally more high-level (cognitively more specific) DSL might be given in terms of a transformation to the low-level DSL or as separate definition, where possibly parts of the definition of the low-level DSL can be reused. The introduction of a cognitively more universal DSL (computationally low-level DSL) requires the implementation of semantics and transformations from existing relatively high-level DSLs. Any granularity of abstraction can be realized this way within the means of object-orientation and constraint languages.

[30]The possibility to represent abstraction by provision of abstraction mechanisms within DSLs is not subject to consideration, since this has been subject to extensive discussion. However it must be noted that internal abstraction mechanisms might in cases match requirements of model-based reasoning best, it is assumed that it does not in general.

Designing DSLs as Types

Much of model-based reasoning appears as the the evaluation of similarities of different models where differences of the perceived structure of systems are evaluated along the resulting behavioral differences that can be observed through experiments (e.g. generalization, limiting case abstraction, idealization, analogical reasoning). Whereas the evaluation of similarities of observations is subject to data aggregation and data visualization methods, the evaluation of structural differences of models is subject to model representation. If economies of representation and interaction apply small conceptual differences should appear as small differences in the perception of a model specification, whereas great conceptual differences should appear as great perceptual differences. For this, representations should directly invoke the cognitive images of mechanisms under consideration while operations that are to be performed mentally and visually on a model should be enabled and represented explicitly. The level of abstraction used for reasoning (base level), thus the properties of imaginable demonstrative mechanisms depend on the subject under consideration. Thus, although model-based reasoning might provide some general hints that support balancing of properties of DSLs (e.g. simplicity vs. expressivity), no specific properties of types, thus DSLs, can be stated against the background of model-based reasoning only. However it is apparent that explicitness of representation is a fundamental requirement and that unnecessary degrees of freedom, e.g. through introduction of generality, may make relevant information (mechanism or modification) implicit, thus lowers adequacy for scientific reasoning, while restriction may prohibit necessary modifications. Further, for evaluation of transparency, DSLs might not be analyzed in isolation, but they must be evaluated in relation to other concepts that are related to the DSL by operations and practices of model-based reasoning. In general, it appears to be a reasonable design goal of a DSL that it matches thought patterns well (i.e. mechanisms), instead of exactly determining of the degrees of freedom (boundary of the class).

Like formal ontologies, DSLs are meant to formally capture the knowledge of a domain. Given the developments of ontologies, it appears that these are rather meant to capture ontologically well-founded knowledge of real systems with little uncertainty, since they are designed to automatically derive specific knowledge about real systems that is not explicitly represented. Thus, it aligns well with systems design and systems analysis, with little ontological uncertainty. In contrast, DSLs for model-based reasoning are rather meant to capture imagined mechanisms for the use of the discovery of knowledge by means of simulation studies, which much more aligns with necessities of cognitive interaction with knowledge representations. However, it is generally a matter of degree to which DSLs might encompass knowledge about systems. Existing developments show that formal ontologies can be integrated technically with metamodel-based language definitions for using inference mechanisms. With all relevant knowledge of system formalized, information - such as state trajectories or optimal designs - may be derived fully by automated reasoning. Uncertainty however requires to take cognitive methodological aspects under considerations, such that technically speaking the domain modeled is not the real system under consideration, but the method for the inference of properties, including the simulator. In this case, abstraction hierarchies are to be designed to reflect cognitive requirements (documentation, anambiguation), that not only must take system knowledge into account, but also the experimental procedures and related scientific discourse. However, the more knowledge is established and formalized, automated reasoning might support the elaboration of system design issues, the system being the real system or in the case of uncertainty

the cognitive system encompassing modelers, simulators and experimental procedures.

Given a high degree of uncertainty, sources of models might be found in type hierarchies itself (analogical reasoning), where the source is an abstract mechanism (e.g. a modeling paradigm). Whereas the type of the source is ontologically founded by being a well accepted abstraction of models, the application of this type in a distant domain particularly requires transparency, since validity is established through evaluation of structural similarities that have to be evaluated in discourse. Existing developments however show that many used high-level modeling paradigms in particular those associated with great uncertainty (e.g. agent-based modeling, cellular automata, see Chapter 4) lack transparency. Explicitness of representation is a fundamental aspect for which both, the concepts of a language and its syntactical appearance account for. Transparency is based on transparent relations to other types, thus each paradigm (DSL) should be explicitly related to other paradigms. Thus, MDE supports the explication of hierarchies of computational and concept abstraction and the composition of paradigms from known paradigms which promotes explication and unambiguation of the semantics of paradigms, in particular, where paradigms are currently defined be monolithic tools. However, when parts of type hierarchies are represented by metamodels, explicit representation requires familiarity of object-oriented modeling of class hierarchies. Further, with MDE it is possible to explicitly consider syntactical aspects of languages. Moreover, it appears that DSLs in the sense of types in a type hierarchy allow to connect ontologically well-accepted knowledge formalized by means of formal ontologies with DSLs that support the method of M&S.

Since paradigms are subject to discussion and convention, it must be expected that there is considerable amount of modification to the paradigm, thus the DSL under discussion, before it establishes a type. Through the gains in efficiency with respect to implementation of language tools, paradigms might find explicit consideration in form of DSLs for which costs of development appeared prohibitive in the past. However, it must be noted that the MDE is relatively recent development, such that aspects such as debugging and profiling are generally not supported to the level at which tools for programming provide support.

The framework presented in Chapters 2 and 3 is applied to cellular automata modeling in EMS as documented in the following chapters. First, the basic properties of cellular automata and respective tools are presented followed by the characteristics of a class of cellular automata that provides the base for the definition of a respective DSL (Environmental Cellular Automata Language, ECAL). Second, a concretization of the approach of MDE for the context of EMS is presented (Chapter 5) and a conforming implementation and applications of ECAL (Chapter 6). Chapter 7 concludes this thesis.

4 Cellular Automata

The field of modeling with Cellular Automata is diverse and a comprehensive discussion is beyond the scope of this thesis. However, a characterization of basic formal and historic aspects explains common general aspects of modeling with Cellular Automata and what distinguishes the application of Cellular Automata in EMS from other applications. This is the basis for the identification of concepts for a modeling language for CA in EMS. In particular, it is shown that different types of Cellular Automata are not to be distinguished by formal properties only, but their characterisation requires the consideration of pragmatic aspects. For this, the basic notions of CA are presented first. Then, CA a are presented in more detail in the context of main fields of investigation: CA as a model for parallel computation, CA for modeling physical processes at the microscopic scale and CA for modeling physical processes at the macroscopic scale as the basis of the definition of ECAL.

4.1 Basic Notions of CA

The first application of Cellular Automata is attributed to John von Neumann who developed this framework in the late 1940s following a suggestion of his colleague Ulam (Vichniac, 1984). Inspired by biological systems and digital computation, the application of the Cellular Automata framework by von Neumann was driven by the investigation of the question: "What kind of organization is sufficient for an automaton to be able to reproduce itself (Mitchell, 1996; Kari, 2005)?". One concrete goal of investigation was the design of simple self-replicating artificial systems that are also computationally universal, in which von Neumann succeeded using the framework of Cellular Automata (Kari, 2005). Since then, the notion of Cellular Automata refers to a modeling paradigm, according to which systems are basically conceptualized as a regular lattice of interacting elements - the cells -, each of which is a simple automaton[1]. What distinguishes CA formally from other more general notions of "iterative networks" or "automata networks" is their homogeneity and local connectivity among cells and a homogeneous update rule across all cells (Mitchell, 1996).

4.1.1 Basic Formal Aspects

The formal definition of the most basic "classic CA" presented in the following has been the base for a variety modifications and extensions:

Formally, a *classic CA* is a discrete-time and discrete-state Dynamical System that is specified as (see Worsch (1999)):

[1] Other terms used are 'Cellular Automaton' and 'Cell Space'. Please note that the term Cellular Automata refers to the modeling paradigm, but also to conceptualizations at more concrete levels of abstraction, such as parameterizable models and models. In this text, the meaning will stated explicitly, if not clear from context.

- A regular *lattice* L of s homogeneous cells, which is typically defined by an Euclidean grid $\mathbb{Z}^d$, with the CA having d dimensions and each cell being referenced by a space coordinate l ($l \in L, L \subseteq \mathbb{Z}^d$).
- A finite number of possible states $\tilde{Q}$ that each cell can have. The set of possible global states Q is composed of *global configurations* (c), where a global configuration c maps a specific state to all cells: $c \in \tilde{Q}^s$, with s being the total number of cells of the CA[2].
- A finite *neighborhood* N: Each cell is assigned a set of m neighboring cells, that is typically modeled by means of a vector of offsets: $N = \{n_1, \ldots, n_m\}$, with $n_i \in \mathbb{Z}^d$. The state of all cells in the neighborhood of a cell at l is the *local configuration* ($c_{loc} \in \tilde{Q}^m$). The vector of neighbors is typically invariant and the same for all cells, however local configurations at the border of the CA require special handling. Thus, the treatment of local configurations might require generalization in form of a *localization function* ($\mu(l, c) : \mathbb{Z}^d \times \tilde{Q}^s \to \tilde{Q}^m$) that returns c_{loc} given a coordinate l and the global configuration c (see below).
- A *local transition function* $\delta^{loc} : \tilde{Q}^m \to \tilde{Q}$ that maps a local configuration to state of the cell under consideration. The global transition function δ results from δ^{loc} by parallel application of δ^{loc} to all cells: $\delta : \tilde{Q}^s \to \tilde{Q}^s$ is defined as $\delta(c) \to c' \Leftrightarrow \forall l \in L : c'_{loc}(l) = \delta^{loc}(\mu(l, c))$.

There exist typical neighborhoods of classic CA that are defined at the base of distance of cells in terms of the index (radius): the *Neumann-neighborhood* as the neighbors within a given radius in capital directions ($N^{Neumann} = \{(o_1, ..., o_d) | \sum_j |o_j| \leq r\}$, with o being the offset in the jth dimension) and *Moore-neighborhood* as neighbors within a given radius in all directions ($N^{Neumann} = \{(o_1, ..., o_d) | \max_j |o_j| \leq r\}$).

Computer simulation of CA restricts the size of lattices to be finite. There exist typical *boundary conditions* that describe how local configurations at the boundary of a lattice are obtained and provide patterns to deal with irregularities of finite grids. Typical boundary conditions are *adiabatic* (addition of virtual cells and duplication of values of boundary cells), *cyclic* (neighbors are taken from the opposite boundary, also called "torus", "periodic"), *mirror* (addition of virtual cells with values that are symmetrically reflected values at the border) and *constant*[3](Worsch, 1999). The local transition function is typically associated with a *rule table*, where each entry maps a possible local configuration to a state of a cell.

4.1.2 Method and Pragmatics

A particular aspect of pragmatics appeared to be a fundamental characteristic of CA-based modeling from the beginnings: von Neumann insisted on the number of states of CA being small, so that high-level functions are not to be explicitly encoded within the local transition functions of the CA, but that high-level organization is the result of simple parallel computations (Mitchell, 1996). A further characterizing feature of CA-based modeling is the duality between Cellular Automata as a model of massive parallel

[2] $\tilde{Q}^s = \tilde{Q}_1 \times \tilde{Q}_2 \times \cdots \times \tilde{Q}_s$.

[3] These boundary conditions can be modeled by means of correspoding localization functions, e.g. $\mu((0, 0), c)((-1, 0)) = c(m, 0)$ (cyclic), where d is the highest coordinate of the first dimension in the lattice (Worsch, 1999).

computation and as a model of real physical systems. Thus, on the one hand, CA-based models have been used to simulate real physical systems, where physical systems are thought to perform computations at a microscopic scale. On the other hand, a major motivation for including aspects of physical systems in CA is the vision to employ micro-scale physical characteristics of real systems for the simulation of CA (according to the notion of "programmable matter"[4]). For direct physical simulation of CA, CA must follow the rules of physics (e.g. reversibility and conservation laws) and be computationally universal (Kari, 2005; Mitchell, 1996). Thus, the investigation of computational properties of CA and basic properties of physical systems is closely related.

4.2 CA for Modeling Parallel Computation

The employment of the notion of CA for the characterization of parallel computation is primarily concerned with issues related to computational universality, synchronization of computations and the inclusion of physical laws (e.g. reversibility, conservation laws). Typical investigations are related to the effort of computations (computational complexity) and the decidability of properties given a CA (e.g. reversibility). Further typical fields of investigation in this context are language recognition, cryptography and fault tolerance. In the following, corresponding relevant findings and pragmatic aspects are shortly characterized.

4.2.1 Universality

Roughly, there are three approaches for the investigation of universality of CA. First, specific CA developed that mimick a specific universal computer one-by-one (e.g. Turing machine). Second, specific CA are designed such that they implement basic logical (OR, XOR etc.) and arithmetic operations that are combined to built up universal computers. Third, "intrinsic universal CA" are identified that are able to simulate any other CA (see (Bandini et al., 2001; Kari, 2005; Mitchell, 1996)). The investigation of those approaches typically includes the manual construction of specific CA followed by a mathematical proof of universality. Depending on the level of generality of discussion, CA encompass specific rules only or specific rules and a specific initial state. Methodically, the investigation of universality is in large parts concerned with proving equivalence of different models in that a CA is able to simulate another model (e.g. CA, Turing machine). Equivalence, in particular in the first two cases, typically refers to that fact that different CA mimick the same high-level conceptualization (e.g. logic gates, Turing machine).

One typical goal of studying universality is investigation of the trade-off between the number of states, the size of the neighborhood, the dimension of the CA and the computational complexity. Although experiences exist on trade-offs between different properties of CA (e.g. trade-offs between the number of dimensions, size of neighborhood, number of states), general results have not been stated (Sarkar, 2000). For example, von Neumann developed a universal two-dimensional CA with Neumann-neighborhood and 29 states that mimicks a Turing machine. This CA has iteratively been reduced to a CA 4 states (Bandini et al., 2001). The universal two-dimensional Game-of-Life CA (see Chapter 4.2.4)

[4]CA have also been used as a mere computational tool to approximate PDE, however most influential works consider CA as a primary conceptual framework that provides the background for reasoning about systems. For this thesis, the latter aspect is of primary interest.

can be used to implement basic logical operations with Moore-neighborhood and 2 states (Bandini et al., 2001). The smallest known intrinsic universal CA has one dimension, two nearest neighbors and 6 states (Kari, 2005).

The probably most influential result of the investigation universality for EMS is that universality is a very common property in CA - a rule rather than an exception -, and that even very simple rules may be computationally universal (Kari, 2005). Moreover, it has been shown that - like many other properties (see below) - the problem of deciding whether a CA is computation-universal based on the local rule is undecidable (Sarkar, 2000).

Other exemplary fields of investigation within the context CA as parallel computers are the investigation of CA as parallel language recognizers and the issue of the synchronization of parallel computations without global signal. These fields of investigation share some methodological properties with investigation of universality: CA are typically perceived as a parallel version of some mechanism that is well-understood in the non-parallel case. Further, CA are typically handcrafted and the investigation typically involves formal proofs of properties: Some relationships of specific types of CA models - e.g. One-way CA (OCA) - and language classes (e.g. OCA recognizes context-free languages) have been proven (Kari, 2005; Sarkar, 2000). From a perspective of pragmatics, it is a characterizing feature that investigations of language recognition are typically cast into higher-level conceptualizations, e.g. language recognizers are described in terms of propagating and intersecting signals, which are given semantics in terms of classic CA (Mitchell, 1996; Kobayashi and Goldstein, 2005; Sarkar, 2000).

In short, the usage of CA as a model of parallel investigation is mainly built upon analytical methods. In contrast to simulation-based investigations, the lattice might be infinite. Further, investigations typically involve the specification of some high-level conceptualization in terms of CA, where high-level conceptualizations are not to be specified directly within the states or transitions, but they emerge from careful considerations of state sets, rules and initial states. Fields of investigation may introduce types of CA models (e.g. OCA, von Neumann CA) which are typically restrictions to the classic CA approach that incorporate specific high-level concepts, that are central part of investigations as they provide the base for measuring equivalences. Whereas simulation-based studies are basically constrained by available resources - thus finite number of cells etc. -, theoretical considerations may use infinite time ($t \to \infty$), infinite lattices, arbitrary numbers of dimensions and are more amenable for the incorporation of non-determinism. In *non-deterministic CA* several different configurations may follow from one configuration (δ is a relation that is not uniquely defined). Non-determinism may take the specific form of asynchronous update (*asynchronous CA*), where at each timestep for each cell, it is decided non-deterministically, if δ^{loc} is being applied. In particular, the investigation of equivalences of deterministic and non-determistic CA (synchronous and asynchronous CA) is of particular interest (e.g. Golze (1978) or Sarkar (2000)). Moreover, it is a typical characteristic of the former fields of investigation that CA models are typically not used for modeling specific quantitative aspects of real systems, but rather target the investigation of general qualitative properties of hypothetical systems.

4.2.2 Reversibility and Conservation of Quantities

Fundamental questions related to CA as a model of parallel computation is the construction of CA that are reversible and conserve quantities. The reversibility - also referred to

as "invertibility" - of global and local transition functions refers to the possibility to find a local or global rule that, if applied from any state on a trajectory of the original rule, produces the original preceding trajectory in reverse order. This requires every configuration to have a unique predecessor. The discussion of reversibility is typically based on the algebraic analysis of injectivity, surjectivity and bijectivity of transition functions, where CA with infinite lattice and spatially periodic CA, and finite CA with a finite number of cells are considered (Kari, 2005). Although some general results have been stated, it is in general, difficult if not impossible to determine the reversibility of a given finite classic CA (Kari, 2005; Sarkar, 2000). However, restricting the properties of CA makes the issue more amenable to analysis, thus properties (e.g. reversibility, dynamical properties, see below) are decidable that are not in the general case (see (Sarkar, 2000; Manzini and Margara, 1999; Dow, 1997). In particular *additive CA* - also referred to as *linear CA* - have been used since they are amenable for analysis and have shown to be able to produce as complex behavior as general CA. Additive CA (linear CA) are characterized through a additive linear local and/or global transition function (e.g. if the local update rule is of the form $\delta_{loc}(a_1, a_2, ..., a_n) = c_1a_1 + c_2a_2 + ... + c_na_n$ for some constants $c_1, c_2, ..., c_n$ and states of the local configuration a_i, Kari (2005)).

Besides reversibility, a CA that obeys basic physical laws must conserve quantities (e.g. energy, momentum, mass etc.). Conservation of quantities might be manually crafted into the rules of CA and proven for the specific rule. Theoretical considerations are typically concerned with determination of conserved quantities by a given CA (Kari, 2005). Here the class of *number-conserving CA* (NCCA) is widely used as the base for investigation. NCCA represent states as numbers where the sum of states over all cells remains constant. This property is used to reason about conservation of quantities (e.g. state may represent the quantity of something in a cell, Moreira (2003)). Durand et al. (2003) proves decidability of number conservation of d-dimensional CA with general boundary conditions, given the number of states and access to a rule table that defines the transition function. Further, it has been shown that NCCA are capable of universal computation (Moreira, 2003).

Although number conservation can be built into CA models manually and proven individually, Margolus (1984) introduced the widely used class of *partitioning CA* that is a variant of CA models that allows straightforward implementation of number conservation and reversibility by construction (Wolfram, 2002). The basic idea of partitioning CA is the usage of *block-rules* as illustrated in Figure 4.1 at the example of the "Margolus neighborhood".

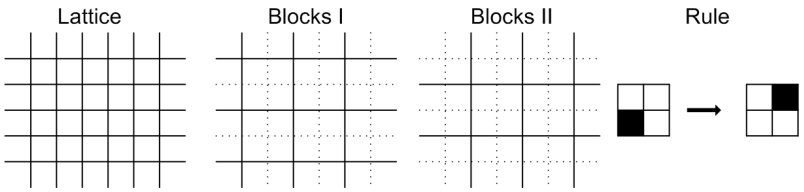

Figure 4.1: Partitioning CA by means of block rules at the example of the Margolus neighborhood. The lattice is divided into non-overlapping sublattices to which block rules are applied cyclically.

The lattice of the CA is divided into different two sublattices that are made up of regular non-overlapping blocks, where each block encompasses a square of four cells. The local transition function defines the state transition for a block in that it prescribes the state

of a block with all contained cells as a function of the state of the cells of the block. The local transition function is applied to all blocks of one sublattice at a time step, where sublattices are altered between time steps (an example of model is given in Chapter 4.2.4).
.

4.2.3 CA as Dynamical Systems

The relation of characteristics of CA and behavioral characteristics of Dynamical Systems (see Chapter 2.2.2) is subject to many research efforts, in particular when properties are present that have been proven to be difficult to represent using classical mathematics (Mitchell, 1996; Wolfram, 2002). In particular, the works of Wolfram (Wolfram (2002) provides exhaustive documentation) promoted this area of investigation, based on the discovery that the most complex type of behavior can be found in even the simplest CA.

Wolfram Classes Wolfram's classification scheme (see Wolfram (1984)) is a qualitative classification scheme that is commonly used to relate Dynamical Systems and CA:

- *W1*: CA tends to spatially homogeneous state.
- *W2*: CA yields a sequence of simple stable or periodic structures.
- *W3*: CA exhibits chaotic aperiodic behavior.
- *W4*: CA yields complicated localized structures, some propagating.

These qualitative behavioral classes - or more precisely defined quantitative interpretations (e.g. Culik and Yu (1988)) - are related to classes of long-term behavior of Dynamical Systems: *W1* corresponds to infinite growth or fixed point behavior, *W2* corresponds to limit cycle behavior and *W3* corresponds to chaotic systems with strange attractors (Wolfram, 1984). *W4* is perceived to exhibit "more complex" behavior in that self-organization (reduction of entropy) takes place that shows itself by the appearance of localized structures (Wolfram, 1984).

In the tradition of seeking simplicity, Wolfram's *elementary CA* have been subject to extensive investigation. Elementary CA are 1-dimensional CA, with binary state and nearest neighbors of radius 1. Although no particular number of cells is associated with elementary CA, it appears that investigations focus on a sizes of magnitude from 10^1 to 10^2 with cyclic boundary conditions (see Wolfram (2002)).

One main result from the investigation of CA as Dynamical Systems is that even the most simple class of elementary CA shows all types of long-term dynamical behavior. However, another result of investigations is that "CA behavior is so complex that almost any question about their long-term behavior is undecidable (Kari, 2005)". This encompasses the undecidability of limit sets, thus Wolfram's or similar classifications (Kari, 2005; Mitchell, 1996).

Simulation-based Investigation and the Edge of Chaos This gives rise to an approach to investigation, that is based on the simulation of CA and subsequent analysis of trajectories in order to understand the relation between structural and behavioral aspects. A number of examples (Langton (1990), Gutowitz (1990), Li et al. (1990), Kayama et al. (1993)) investigate the relationship between statistical measures of CA rules and long-term

behavior. Whereas Gutowitz (1990) aims at approximately predicting statistical properties of CA behavior based on a statistical measure of rules (mean-field approximation), Langton (1990); Li et al. (1990); Kayama et al. (1993) relate statistical measure of rules (e.g. λ as share of non-quiescent rules) to Wolfram classes and computational universality. Although there are differences, investigations suggest that there is an ordering in rules with respect to statistical measures on rules. For example Langton (1990) and Li et al. (1990) argue that the ordered variation of λ by means of the respective variation of CA rules corresponds to the CA transitioning through long-term behavioral regimes (phases) from completely ordered (fixed point) over periodic to unordered (chaotic), however the investigation of these findings is subject of ongoing research. Complex behavior, identified by the appearance of localized structures (class W4) appears around a "critical value" of λ that is referred to as "the edge of chaos". Universal computation is perceived to be likely at the edge of chaos, since computation requires some facility to store and transport information that is given by moving structures as signals (Langton, 1990).

Like research on CA as models of parallel computation, the investigations of CA as Dynamical Systems take place at a relative level of generality, where rather qualitative properties of systems are under consideration. However, applications to real systems and investigation of more specific quantitative properties appear to be inspired by the possibility of even the most simple CA to show emerging complex behavior along with simple causal structures (see Chapter 4.3). Although the CA models are similarly simple with respect to the sets of possible states and neighborhood, investigations are largely based on computer simulation and investigation of observed trajectories. Experiments typically aim at exhaustive exploration of rule spaces, that are defined by the dimension of CA, sets of states and neighborhood. Moreover, experiments encompass the use of random initial conditions, that is typically handled within Monte-Carlo experiments[5]. Where exhaustive exploration is not possible, a sampling of rules reduces the number of simulations within an experiment series (see Wolfram (2002); Gutowitz (1990)).

4.2.4 Exemplary CA models

For illustrative purpose and to provide a concrete idea of classic CA, this chapter presents some exemplary CA models that are used within this field of investigation. Three famous exemplars of for Cellular Automata that have extensively been analyzed with respect to computational properties are Wolfram's *Rule 110* CA, Conway's *Game-Of-Life* CA (GOL) and Margolus' *Billiard-Ball Model* (BBM) CA. All belong to Wolfram Class IV and have been proven to be computationally universal.

Rule 110

The rule 110 CA is an *elementary CA* with two nearest neighbors and two states. Figure 4.2 (a) presents the local transition function in a typical graphical notation of rules of elementary CA. The first line denotes the local configurations and the second line the corresponding next state of the cell under consideration (Wolfram, 2002). An exemplary trajectory if is given by Figure 4.2 (right) illustrating the occurrence of an at first seemingly chaotic behavior that is followed by the appearance of localized structures - appearing as big triangles and diagonal patters against a background of small triangles - from random

[5]Monte-Carlo refers to randomly generated initial states.

initial conditions[6].

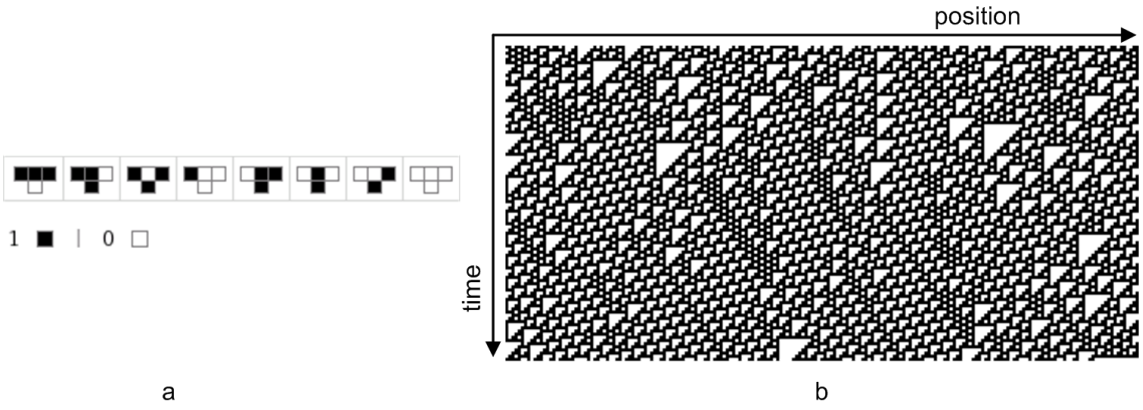

Figure 4.2: Elementary Cellular Automaton Wolfram rule 110: transition rule (a) and exemplary trajectory with random initial conditions (generated by http://www.wolframalpha.com).

By interpreting localized structures as moving gliders that interact in a predictable manner (e.g. annihilation, production of new gliders) Cook (2004) proves universality of Rule 110 by showing that these interacting localized structures can be designed such that the Rule 110 CA can emulate a "cyclic tag system", which is proven to be computationally universal. Thus, a number of properties, e.g. long-term behavior, cannot be decided for this simple rule.

Game-of-Life (GOL)

The probably most famous Cellular Automaton is *Conway's Game of Life* (GOL) that has been developed with the motivation to construct a rule that is basically not predictable (Gardner, 1970). It is a 2-dimensional binary CA with a Moore neighborhood, with each cell either being the state "alive" or "dead" and a prototypical example of the class of "counting rule CA", where the next state depends on the count of neighbors being in a particular state. Such CA are widely applied in modeling physical processes (see Chapter 4.3):

- A "dead" cell with three "alive" neighbors becomes "alive" (birth).
- An "alive" cell with less than two "alive" neighbors becomes "dead" (isolation).
- An "alive" cell with two or three "alive" neighbors remains "alive" (survival).
- An "alive" cell with more than three "alive" neighbors becomes "dead" (overpopulation).

Figure 4.3 presents some exemplars of local structures[7] that are those characteristic of features of GOL that have been subject to intensive research (Kari, 2005):

[6] The trajectory in generated by WolframAlpha `http://www.wolframalpha.com/input/?i=rule+110&lk=1&a=ClashPrefs_*MathWorld.Rule110-`.

[7] Examples are taken from `http://en.wikipedia.org/wiki/Conway's_Game_of_Life`. See this page for more examples and animated pictures.

- Still life: Structures that remain fixed (a).
- Oscillator: Structures that oscillate at a fixed location (b).
- Gliders (spaceships): Structures that move through the lattice (c).
- Glider guns: Oscillators that emit gliders (d).

These types of structures can be used to (theoretically) construct logic gates and counters which enables the construction of computationally universal computer, although it originally draws from analogies of the "rise, fall and alternations of a society of living organisms [...] (Gardner, 1970)." .

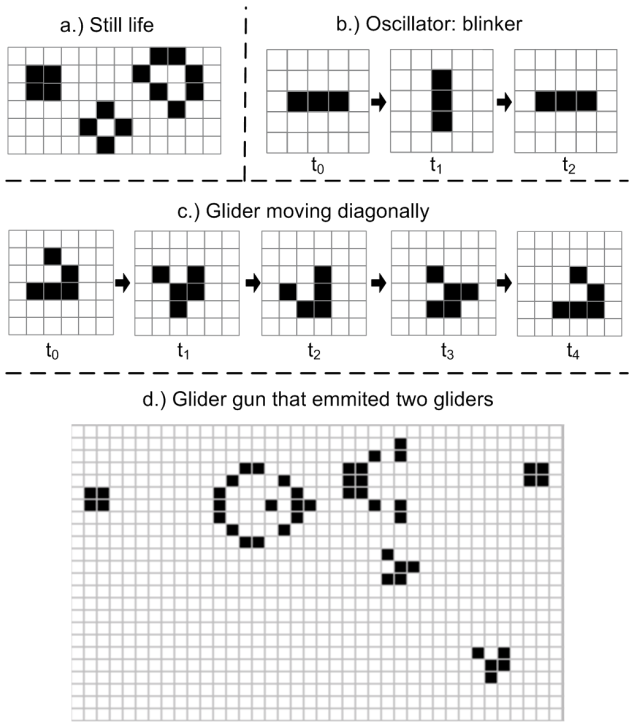

Figure 4.3: Examples of different types of local structures in GOL: still life (a), oscillator (b), glider (c) and glider gun (d).

Billiard-ball Computer: BBM

The "billard-ball computer" is model of a reversible computer, where the basic building blocks are colliding "billiard balls" that can be arranged such they build reversible logic gates that can make up a universal computer (Margolus, 1984). Margolus (1984) introduced a reversible CA model of the billiard ball computer, along with the type of "partitioning CA" (see Chapter 4.2.2). Figure 4.4 presents the rules of the BBM model at the base of the Margolus-neighborhood.

The two-dimensional "billiard-ball computer" is a reversible CA that has shown to be universal (Kari, 2005).

Although interesting from qualitative point of view, both Rule 110 and GOL are exemplary in that they are not applied to real systems in order to derive quantitative statements

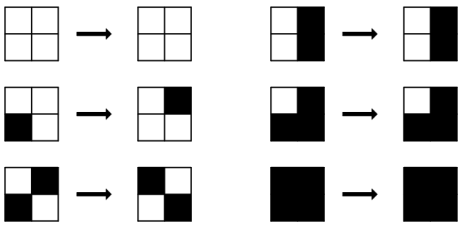

Figure 4.4: The rules of the BBM model using the Margolus-neighborhood.

about them. Instead these models exist at a very high level of generality in that they are basically used to investigate fundamentals of parallel computation and basic qualitative properties of organization and complexity in nature itself. Although this also applies to the BBM model, it has been shown in Toffoli and Margolus (1986) that the BBM model can be seen as a slight variation of the HPP lattice gas automaton that has been applied to modeling real systems (see Chapter 4.3.3). Also these examples represent the methodological approaches to CA modeling in these fields, whereas GOL and BBM are carefully constructed CA models which are widely treated analytically. The interest in Rule 110 appeared as a result of exhaustive exploration of the rule space of elementary CA by means of digital simulation.

4.2.5 Method and Pragmatics

The commonality of the field of investigation of CA as model of parallel computation and Dynamical Systems is that the aim of simplicity of universal CA lead to the use of simple CA with relatively few states and small neighborhood. However, there is a variety of modifications to the notion of Cellular Automata. A restrictive paradigm allows analysis and enforces comparability (e.g. linear CA). However, there are extensions not covered in the above overview, e.g. infinite lattices, non-deterministic rules and asynchronous update of cells, that go beyond the notion of classic CA (see Worsch (2009)). The undecidability of interesting properties of CA and the resulting unpredictability of behavior in non-trivial cases requires an experimental approach to investigation with digital simulation, when behavioral properties are under consideration. Often, small-scale properties of CA are of less interest than the emergent aggregate properties (e.g. signal propagation and processing) in that implementation by means of CA merely shows the possibility of some emergent functionality under consideration (e.g. language recognition).

The main goal to research of CA as a model of parallel computation in theoretical computer science is to characterize CA with respect to general characteristics of computation (universality, complexity etc.). Some CA (e.g. self-reproducing automata) are typically set up manually with states and rules carefully chosen and desired properties proven analytically. Other works relate general classes of CA, in particular when equivalences in terms of simulation are investigated. Results are typically derived analytically without simulation. Models aim at minimality in terms of small set of states and computational complexity given a desired property. In terms of pragmatics, issues of decidability fall into the same category of investigation.

CA models are typically discrete with a state set at the order of 10^1 and nearest-neighbor Moore or Neumann neighborhoods. Whereas, theoretical considerations widely use infinities (e.g. time, number of cells) and non-determinism, the finite resources of

simulation-based approaches typically limit dimensions to 1 or 2 and use nearest-neighbor Neumann or Moore neighborhood with cyclic boundary conditions in order to simulate an infinite lattice. However, simulations are basically bound to finite time, finite lattice and finite number of dimensions - typically $d \leq 3$. The relation to real systems is rather made at an abstract qualitative level, rather than rigorously quantitative. A frequently re-ocurring pragmatic aspect is the usage of the notion "particles" as a higher-level abstraction for items that transport information ("signal") across CA. Particles and their behavior might be directly represented within the state (e.g. 0 means "particle present" and 1 means "particle not present") or they are localized structures composed of cells with particular states (e.g. gliders).

Particularly the works of Wolfram promoted simulation-based approaches with investigations of CA as Dynamical Systems with the investigation of "rule spaces" at the base of simulation. Exhaustive simulation with traversal of complete rule spaces is applied in order to relate behavioral patterns with characteristics of rules. Where exhaustive simulation is intractable, more sophisticated methods of traversal of the rule space have been developed that reduce the number of simulations. Wolfram (2002) however points out that reductions, either of the number of experiments or the compression of data (e.g. statistical aggregation), introduce assumptions, the results of which are typically unknown under epistemic uncertainty. Further, Wolfram (2002) highlights the importance of tool support and pragmatic aspects, in that tools not only facilitate the necessary simulation of CA models, but also they provide necessary analysis facilities, where the visualization of data is a fundamental aspect, in particular at early stages of research, where intuition is built up and hypothesises are stated: "Yet what I have found is that if one manages to present this data [trajectories] in the form of pictures then it effectively becomes possible to analyze very quickly just with one's eyes. And indeed, in my experience it is typically much easier to recognize unexpected phenomena in this way than by using any kind of automated procedure for data analysis (Wolfram, 2002)". This points to the important aspect of CA-based modeling that frequently occurs in all fields of investigation that CA are used to build up intuitions within simulation-based studies.

The presented CA used in simulation-based investigations widely share computational characteristics of those CA used in modeling physical processes at the microscopic scale. These CA and the characteristics of corresponding tools are described in the following chapter. Further, micro-scale CA provide prototypical mechanisms as templates for macro-scale CA used in EMS.

4.3 CA for Micro-Scale Modeling Physical Processes

A basic field of investigation of using CA is physical systems at the micro-scale, where it is considered an alternative to the use of the traditional approach based on PDE.

4.3.1 Relating Scales with CA

Besides the use of CA as a numerical method of approximation and the interpretation of CA as Dynamical Systems (see Chapter 4.2.3), Vichniac identifies the usage of CA as an original modeling paradigm as the approach with greatest depth of ambition. CA are used as original conceptual framework for reasoning about the structure of specific real systems in that CA models not only reproduce general, rather qualitative, properties

of real systems, but that specific quantitative observations are to be reconstructed and explained by means of CA-based models.

Although the field is generally diverse, it appears that the most fundamental motivation is the aim is to find simple explanations[8] for complex phenomena, in particular for those where traditional continuum approaches, in particular statistical mechanics, and the usage of Differential Equations are associated with fundamental and practical intricacies. The typical approach is to cast assumed micro-scale mechanisms of physical systems into the CA framework and relate these to existing macro-scale descriptions that typically have the form of equation-based physical laws (e.g. conservation laws) and derived equation-based models for specific types of systems (e.g. hydrodynamic systems, see below). The micro-scale typically refers to processes described roughly at the scale of molecules as opposed to macro-scale descriptions that are usually built through averaging micro-scale phenomenology. Further, CA are typically used to find explanations for the behavior of physical systems for which mere descriptive phenomenological characteristics have been stated ("empirical laws") at the macro-scale (e.g. pattern formation, fractal growth).

There exists a variety of competing CA-based models for a variety of application areas. However, literature suggests that there are prototypical physical processes and associated respective prototypical CA-based conceptualizations that explain the essence of CA-based modeling at the micro scale and which are the basis for specializations to particular physical issues and adaptions in other domains, such as EMS.

4.3.2 Prototypical Processes and Phenomena Modeled with CA

In the following the most prominent and prototypical physical phenomena that are associated with CA-based investigations are shortly characterized: fluid flow, diffusion, reaction-diffusion, growth and (self-organized) criticality. CA-based mechanisms modeling these processes can be considered as prototypical mechanisms that have been applied in EMS.

Fluid flow

Fluid flow is associated with flow of matter in a medium that is governed by hydraulic laws[9]. The characterization of flow and associated processes of transport, dispersion, displacement and mixing of fluids is typically subject of scientific studies. Theoretically, the conservation of momentum and mass is the fundamental theoretical constraint of models. The properties of medium and fluid (e.g. geometry, viscosity) are associated with models of specific systems. At the macro-level, hydraulic flow is basically described by the non-linear Navier-Stokes differential equations[10], the solution of which typically requires numerical treatment (Sahimi, 1993). Issues are generally related to non-linearity in combination with complex boundary conditions and interactions at the interface between fluids (multi-phase flow) or fluid and medium (e.g. e.g. flow past obstacles, percolation/flow in porous medium) with many interacting forces (e.g. buoyancy, viscous and capillary forces) making analytical and numerical treatment demanding and often practically intractable

[8] Motivation typically is described by Occam's razor or the quotation: "Make things as simple as possible, but not simpler (Albert Einstein)."

[9] Please note that "fluid" and "medium" may denote anything that follows the respective laws (fluid: liquid, vapor, heat flux, electric current, infection, solar system, medium: pore space, fluid phases of an interspersion, an array of trees, universe etc., Sahimi (1993)).

[10] For non-compressible fluids: $\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho} + v\nabla^2 u$, with u denoting the velocity field, ρ the density, v the viscosity, ∇ the gradient, ∇^2 the Laplacian.

(Sahimi, 1993)). A more fundamental issue of Navier-Stokes (Darcy's Law in case of porous medium) is related to the fact that in continuum equation-based models micro-scale effects are typically averaged. This prohibits inclusion of averaged-out effects and may render it impossible to derive adequate parameters representing the system (e.g. at the presence of capillar effects, small and heterogenuous pores in porous medium, Sahimi (1993)). For some types of flow - e.g. two-phase flow in porous medium - even little is known about governing laws (Sahimi, 1993). CA models (see below) have shown to be particularly useful, where classical, continuum equation-based approaches have limited applicability.

Diffusion and Growth

Diffusion is a fundamental physical transport process of matter that is dominated by the microscopic characteristics of matter under the influence of noise (thermal noise, Brownian motion). The fundamental idea of diffusion is that the "particles" that make up matter move randomly, where the disposition to movement depends on a given level of external noise (e.g. heat). Theoretically, diffusion is basically constrained by the conservation of mass and at the macro-scale described by Fick's Law of Diffusion[11] that relates density (chem.: concentration) and movement of matter (Chopard and Droz, 1998). Generalizations and specializations are specified by other laws (e.g. Chapman-Enskog, Telegraphists equation). Generally, respective models suffer from difficult analytical and numerical treatment, in particular when diffusion appears in inhomogeneous medium, rough (fractal) interfaces between media (Chopard and Droz, 1998). Physical diffusion-driven growth processes appear to be particularly demanding. These are processes with a moving diffusion front, where other processes, such as aggregation, deposition and chemical reactions take place. Although phenomenology of many diffusion-driven processes appears to be well-described through empirical "laws" (e.g. dendritic grwoth, self-similarity/fractals) the exact workings of many phenomena appears to retract from analysis. Such growth processes have in common that they are typically far-from equilibrium and sensitive to local fluctuations. They appear in open systems with input and output (e.g. energy, mass) and the theoretical background from equilibrium statistical mechanics has shown not to be explain many phenomena. Such processes are typically associated with non-linearity, chaotic, self-organized and critical behavior (see below), generally rendering numerical approaches difficult.

Diffusion-limited Aggregation (DLA) is a basic prototypical physical growth mechanism that is based on particles diffusing within a medium and eventually sticking together to form an aggregate. DLA is typically governed through an external spatial field (e.g. electric field, temperature, density etc.). Although, fundamental characteristics of the single processes that form DLA are known, DLA is "not described theoretically by first principles only (Chopard and Droz, 1998)" and spatial fluctuations (small scale variations) play a major role (Chopard and Droz, 1998). Such fluctuations are generally difficult to include in macro-scale continuum analysis (Chopard and Droz, 1998) and DLA particularly retracts from analysis, thus the characterization of systems in is in practice solely based on simulation (Sander, 2000)

Some CA models (see below) are perceived as providing a natural way to include fluctuations (Chopard and Droz, 1998). CA models have shown to reproduce observed char-

[11] Fick's Law of diffusion: $\frac{\partial \rho}{\partial t} = D\nabla^2\rho$, with D being the diffusion constant, ρ the density/concentration and ∇ the gradient (Chopard and Droz, 1998).

acteristics such as growth rates and fractal shapes (Chopard and Droz, 1998). Thus CA models provide a hypothetical generative mechanism for fractal growth processes that can be observed in many areas (e.g. electrodeposition, snow flakes, coast formation etc.). Different diffusion-driven growth processes, such as deposition (aggregation at a surface) or adsorbtion (aggregation, where sticking requires a "free slot") can be modeled as variants of DLA models (see below, Chopard and Droz (1998)).

Reaction and Reaction-Diffusion Processes

Reaction processes are processes where different chemical species react and built resulting chemical species. Reactions are typically described by chemical equations of the form $A + B \xrightarrow{K} C$, that basically give the relation of the different molecules of different species (A, B and C) and the reaction rate (K) of the reaction. Reaction-Diffusion processes combine diffusion and reaction, which is typically described by macro-scale rate equations of the form $\partial\rho_A/\partial t = D\nabla^2\rho_A - K\rho_A\rho_B$ (D being the diffusion constant, ρ density/concentration, ∇ the gradient and K the reaction constant). In general, reaction and reaction-diffusion processes are typically non-linear and many applications relate to far-from-equilibrium systems (open systems) generally rendering PDE-based continuum approaches analytically and numerically difficult, in particular when complex boundary conditions, e.g. heterogeneous medium, are included. Particularly, processes of pattern formation appear to retract from classical modeling approaches and may show anomalous kinetics that conflict with the macro-scale framework. Fluctuations appear to be influential, for which mean-field continuum approaches do not account for (Chopard and Droz, 1998). Various reaction-diffusion processes are documented, for which basic chemical processes are known and empirical laws exist as compact representations of observations, which however pose particular problems when dealing with traditional continuum approaches for deeper analysis.

A famous well-researched example is the formation of Liesegang patterns where a reaction-diffusion process results in the formation of persistent spatial patterns that are formed by a moving reaction front (bands, rings, spirals). Although the exact workings are subject to scientific discussion, some empirical laws have been produced to describe the phenomenology of the formation of bands (spacing law, time law, width law, Matalon-Packter law, see Jahnke and Kantelhardt (2008)). Current research is concerned with understanding and finally controlling the pattern formation (e.g. for nano-scale engineering). Therefore, Liesegang pattern formation is perceived as a Reaction-Diffusion process (propagation) with nucleation and growth (precipitation), where propagation is driven by diffusion and aggregation (nucleation and precipitation) depends on local fluctuations of densities (Chopard and Droz, 1998). There are CA-based models that include micro-scale fluctuations (LGA and Ising) Jahnke and Kantelhardt (2008); Chopard and Droz (1998) and that reproduce some other features of Liesegang patterns (see below).

Another prototypical example of pattern formation and self-organization in Reaction-Diffusion systems is the Belousov-Zhabotinsky reaction in excitable medium. Belousov-Zhabotinsky is a prototypical exemplar of far-from-equilibrium autocatalytic reactions, where one of the reactants is also the result of the reaction. The reaction shows oscillations of concentrations of reactants and periodic propagation of concentration waves (Karapiperis, 1997). Simple CA models (e.g. Greenberg-Hastings, cyclic state, see below) have shown to reproduce spatial patterns.

A further prototyical Reaction-Diffusion process follows the idea that two chemical

species with different speeds interact such that one species is *inhibitor* and the other the *activator* of a chemical reaction (e.g. Schnakenberg model). Inhibitor-Activator models may exhibit chaotic and self-organizing behavior where parameter variation may cause bifurkation from homogeneous state to periodic configurations or stable heterogeneous patterns (Turing patters). The simple Schnakenberg [12] model produces such complex spatial patterns, where concentrations of A and B are kept constant by input. Fluctuations may cause the reorganization of patterns Karapiperis (1997)). Studying the circumstances of pattern formation is done by CA (Karapiperis, 1997), where the inclusion of fluctuations allows better characterization of chaotic systems around bifurkation points (Turing instability in Schnakenberg model, see Karapiperis (1997)). The heterogeneity or the movement of surfaces/interfaces may play crucial role for system behavior of reactive systems (porous media, two-phase flow) and CA allow an intuitive transparent formulation of these phenomena (Karapiperis, 1997).

Criticality, Self-organized Criticality and Fractals

Criticality is a phenomenon that may occur in the above mentioned processes and which appears to by a phenomenon that is particularly amenable to CA-based modeling. Criticality is a phenomenon occurring at phase transitions and associated with a critical temperature T_c, typically associated with "interesting physics (Creutz, 1996)", such as the divergence of correlation length (self-organization) and scale invariance of events (fractal patterns). A phase (e.g. gas - fluid - solid; ferromagnetic - paramagnetic, laminar - turbulent) may be defined by having particular qualitative characteristics. At phase transitions physical observations (e.g. correlation length) diverge or diminish and thermodynamic functions are not differentiable/analyzable. Although the idea of of phase transition is specific to systems with critical temperature, the concept is applied to other fields, where temperature is not the relevant parameter (e.g. directed percolation, turbulent and laminar flow (Chaté and Manneville, 1990).).

Whereas there is a well-defined mathematical framework for characterizing phase transitions between different equilibrium states (renormalization group: universality classes, critical exponents, fractal behavior and ergodicity breaking), phase transitions in non-equilibrium (open) systems between steady states lack general theory and investigation is based on simulation and a typical application for CA (Chopard and Droz, 1998). The typical example of critical behavior and self-organization is that of magnetization, where a system with a temperature above T_c is unordered, thus paramagnetic with no relevant magnetization observed at the macro-scale. Around the critical temperature T_c the system organizes itself such that homogeneous areas (clusters) are formed with the effect of magnetization at the macro-scale. Below T_c the correlation length reaches the size of the system (completely organized). Another example of critical behavior is found in directed percolation (flow in porous medium) where the threshold parameter is the microscopic connectivity (P_c, percolation threshold) of the sites of the medium. At the critical value of P_c the medium changes between permeable (correlation length is size of the system) and impermeable (correlation length $<$ size of the system, Chaté and Manneville (1990)).

In view of lack of adequate theoretical background, investigations aim at characterizing non-equilibrium phase transitions analogous to the equilibrium case (e.g. identify universality classes, critical exponents etc.) and indeed characterize specific systems. In

[12] Schnakenberg model: $A \xrightarrow{k_1} X$, $X \xrightarrow{k_2} \emptyset$, $2X + Y \xrightarrow{k_3} 3X$, $B \xrightarrow{k_4} Y$

theory, two extreme mechanisms of phase transitions are distinguished. First, the system is instable through external influence and small perturbations cause a system-wide phase transition (spinodal decomposition). Second, the system is in metastability (locally stable point) and a relatively great fluctuation causes instability with local nucleation and growth of regions of a phase within regions of the other phase. Since it is possible to bring metastable systems in arbitrary small proximity to instability, the boarder between instability and metastability is continuous (Chaté and Manneville, 1990). Both, small perturbations and nucleation and growth phenomena are particularly amenable to CA-based modeling (see below).

Whereas modeling critical behavior is a matter of tuning respective parameters (e.g. T_c, P_c), Self-Organized Criticality (SOC) is a phenomenon where a system shows critical behavior irrespective of parameter values (e.g. critical temperature T_c). Mathematically, in contrast to critical systems, the critical point of a SOC system, where critical behavior occurs, is an attractor of the system (Bak et al., 1987). SOC are meant to provide an explanation for systems that show fractal behavior in the sense that interesting observations of physical quantities show power-law frequency-size distribution, thus which are self-similar across scales ($N \approx A^{-\alpha}$, where N is count, A is the size and α is constant with a value ≈ 1, see Bak et al. (1987)). Examples of phenomena with power-law distributions are earthquakes, avalanches, forest fire, landform evolution, the development of river networks, for which SOC might provide a possible generative prototypical mechanism. SOC is a complement to chaotic systems that exhibit a variety of behaviors (fixed points) with few degrees of freedom, whereas SOC systems exhibit common features with a great degree of freedom (Creutz, 1996). However, there is evidence that a system with SOC is at the "edge of chaos" (Turcotte, 1999).

A basic ingredient of modeling, thus existing theory, is that an SOC system is perceived to develop into a "minimally stable state" in which small perturbations might lead to effects at all scales which is typically associated with the propagation of perturbations through threshold dynamics (Creutz, 1996; Bak et al., 1987). Two types of SOC models appear: stochastic models in a deterministic environment (e.g. sandpile) and deterministic models in a stochastic environment (e.g. Bak-Sneppen slider-block model, Frigg (2003)). CA models have been developed that show SOC behavior and may encompass respective causal mechanisms. A number of CA models have been developed that obtain SOC with a physical interpretation (see below).

4.3.3 Prototypical CA models

For each of the above mentioned processes exist a variety of CA-based models. Literature suggests that there are prototypical CA-models as the basis of extensions and specializations that model interesting aspects the different types of processes under consideration.

Lattice Gas Automaton (LGA)

A particularly well-developed and widely discussed type of CA is that of Lattice Gas Automata (LGA). According to LGA, a system is defined on a lattice where particles are present (with a velocity) and move across the lattice according to a rule that is applied in parallel for all sites of the lattice. The dynamics of particles is conceptualized as a two-step-process where particles propagate to neighboring sites in the first step (*propagation*) and collide in the second step (*collision*), given that a number of particles enter the same

site from different directions. Rules are such that the *exclusion principle* holds, which states that only one particle can move into one direction at a given site (Chopard and Droz, 1998).

Three types of LGA models are distinguished (Chopard and Droz, 1998):

- *Boolean CA* models, with binary variables indicating if a particle is present moving in the respective direction (one variable for each possible direction).
- *Multiparticle models*, where state is an integer modeling the number of particles present. The exclusion principle does not hold (several particles might move in the same direction).
- *Lattice Boltzmann Method* models, where state is a real number modeling the probability of a particle being present or the average number of particles, that is specified according to the Boltzmann equation which is to be derived for specific models of particle behavior [13].

Boolean CA The HPP and FHP models are prototypical examples of boolean CA LGA models of flow phenomena. The HPP-model was originally developed without reference to CA as a theoretical model to study general statistical properties of systems. It has later been perceived as a classic CA model for modeling specific systems (Chopard and Droz, 1998). HPP is perceived to follow the physical intuition mimicking the actual movement of molecules that in effect models fluid flow that follows the respective macro-scale laws by explicitly including respective physical properties in local rules: local conservation of mass and momentum and invariance under time reversal (Sahimi, 1993; Chopard and Droz, 1998). "Invariance under time reversal" means that, if all directions of particle motions are reversed, the HPP traces back its own history (Chopard and Droz, 1998).

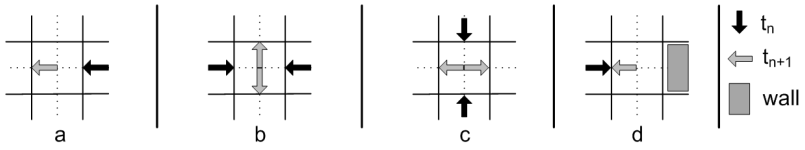

Figure 4.5: Illustration of the transition rule of the HPP LGA model.

Figure 4.5 illustrates the HPP model as a boolean CA LGA with a Neumann neighborhood, where particles move one cell per step on a square lattice along four cardinal directions (Chopard and Droz, 1998). There is ballistic movement until collisions occur (a). At head-on collision, particles are deflected in perpendicular directions (b, c). Particles move on transparently - without collision - otherwise. Boundaries between media (e.g. reflecting walls) can be implemented with relative ease by introducing a cell state that denotes the other medium (e.g. wall) and formulate the behavior of particles at their neighborhood (e.g. reflection, Figure 4.5 (d)). Please note, that Figure 4.5 does not give those rules that result from rotation of rule *a* or other rules that specify simple collisionless motion. Indeed, the introduction of further states requires the addition of additional

[13] The Boltzmann equation is a balance equation which expresses how the average number of particles with a given velocity changes between $(t + dt, \vec{r} + d\vec{r})$ and $(t, \vec{r})$, due to inter-particle interactions and ballistic motion; t and $\vec{r}$ are the time and space coordinates, respectively (Chopard and Droz, 1998).

lattices in the framework of boolean CA, which is referred to as *multiple lattice* CA or the state of a cell may take more than two values such that the model remains a *single lattice* model.

HPP models have shown to reproduce phenomena of sound wave propagation and optics, including the isotropic propagation of sound waves, where the lattice gas represents a medium and a perturbation (wave) is encoded into initial state. Refraction and reflection can be introduced as variants with slightly modified versions of transition rules, where specific cells are designated as reflectors (e.g wall) or refractors (e.g. a lens), e.g. by assigning a corresponding state to these cells. Transition rules are modified such that motion is different within designated cells (e.g. slow motion in a refractor by moving every second step only) or take designated cells in the neighborhood into account (e.g. reflect particles from neighboring reflector cells, see Toffoli and Margolus (1986)). Figure 4.6 illustrates a simulation of wave reflection (from Toffoli and Margolus (1986)) with a concave reflector. Please note, that any shape of the reflector can be modeled in a straightforward way constrained only by the "resolution" of the CA.

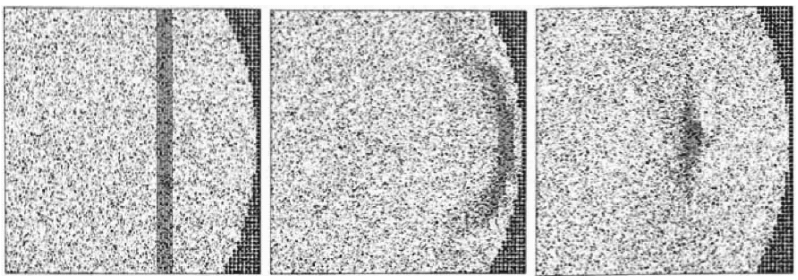

Figure 4.6: A HPP model of a plane pulse traveling towards a concave mirror (left), right after reflection (middle) and approaching the focal point (c, from Toffoli and Margolus (1986)).

It has been shown that the Navier-Stokes equations that describe the flow of fluids at the macro-scale can be derived from the HPP-model. However, the square lattice causes anisotropy[14]. This motivated the formulation of the isotropic *FHP model*, that is an LGA CA based on a hexagonal lattice with six possible directions of movement and six possible particles per site (Toffoli and Margolus, 1986).

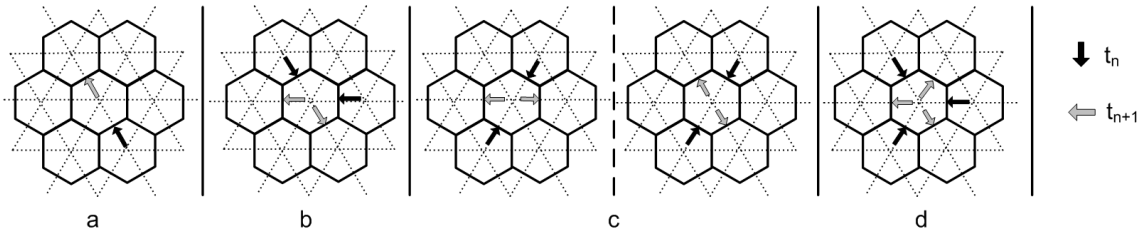

Figure 4.7: Illustration of the transition rule of the FHP CA model.

Figure 4.7 illustrates the FHP model as presented in Frisch et al. (1986) with ballistic motion of particles without collision when particles are not moving head-on (a, b),

[14]Isotropy is the property of a physical system to be invariant under rotations of the coordinate system. Thus, in an anisotropic model, there is dependence of the characteristic of a process on direction.

a three-body collision (d) and two alternative two-body collisions, with each deflection having the same probability. Please note that the rule in Figure 4.7 is not complete as it does not include all collisionless rules and rules that follow from rotation. The FHP can be implemented as probabilistic model (changing directions at 2-particle collisions) or alternating directions can be chosen according to time or a flip bit (Chopard and Droz, 1998). Alternating/probabilistic rules for 2-particle head-on collisions are necessary for isotropic behavior, three-particle collisions are necessary to avoid spurious conservations, such as conservation of number of pairs moving in the same direction (Frisch et al., 1987). Different versions of FHP have been developed with additional rules for a "rest particle" (FHP-II) that efficiently avoids spurious conservations and FHP-III with an exhaustive set of collision rules (Frisch et al., 1987). From the FHP-model it is possible to analytically derive the Navier-Stokes equations and other properties prescribed by statistical mechanics (e.g. conservation) analytically (Chopard and Droz, 1998; Frisch et al., 1987). Further, as HPP, FHP may be used to reproduce the behavior of sound waves (Frisch et al. (1987), see Figure 4.6 . The microscopic scale of modeling shows by the "viscosity" - a parameter that has to be set externally at the macro-scale - following from the structure of the CA (Frisch et al. (1987)). However, fluid flows with high Reynolds number, thus relatively great amenability to turbulent flow, would require intractable numbers of cells in simulations (Frisch et al., 1987). One perceived strength of LGA for fluid flow is the relative ease with which complex boundary conditions, e.g. fractal interfaces, can be included. Figure 4.8 illustrates the geometrical configuration of a porous medium (a) that through which flow can be simulated when adequate boundary conditions and corresponding rules are specified, such as (Figure 4.8 b, c, d) for an FHP model for fluid flow.

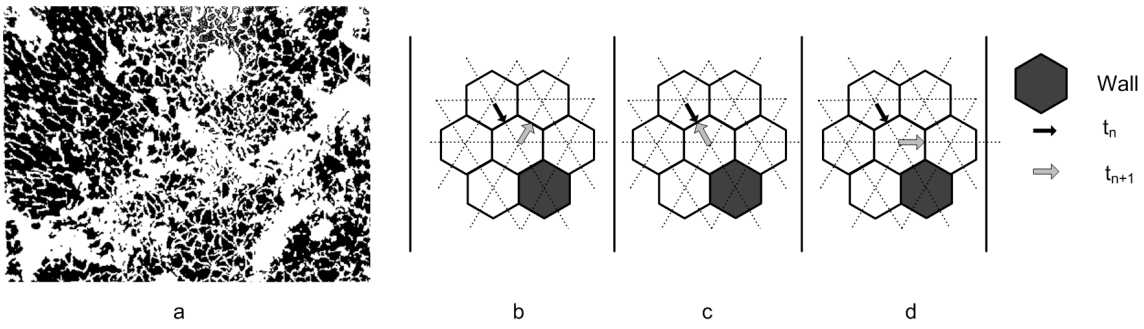

Figure 4.8: Example of a porous medium for which fluid flow has been simulated with LGA in Chen et al. (1991) (a), a corresponding rule for the fluid-solid interface for a FHP LGA (from Chopard and Droz (1998)) with specular reflection (b), bounce back (c) and trapping wall condition (d).

The process of diffusion can be modeled in a straightforward way by means of a probabilistic boolean LGA, that directly reflects the common conceptualization of diffusion as a random walk of particles that preserves the particle number, thus mass. This can be achieved by randomly shuffling the direction of motion of particles at each application of the transition function. Figure 4.9 shows a simple transition rule for four particles entering a cell, according to which the direction of particle movement is randomly shuffled in a cyclic manner (see Figure 4.9, Chopard and Droz (1998)). Rules for less than four particles entering a cell are obtained by simply removing arrows from the rule. Probabilities are constrained by $p_1 = p_3 = p$ (counterclockwise and clockwise reflection) and

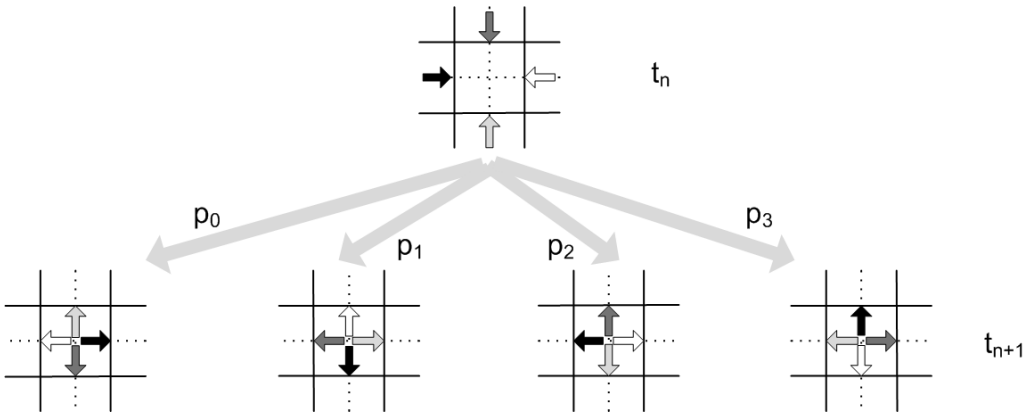

Figure 4.9: Transition rule of a probabilistic boolean CA LGA for diffusion with cyclic random rotation of the direction of movement (from Chopard and Droz (1998)).

$p_0 + 2p + p_2 = 1$ (p_0 is no reflection, p_2 is reflection, see Chopard and Droz (1998)). When scale is chosen sufficiently small, the diffusion CA obeys the law of diffusion and shows isotropic behavior at the macro scale, despite square lattice (Toffoli and Margolus, 1986). Boundaries between media can be modeled analogously to LGA fluid model. Further, the diffusion constant is adjustable: by choosing p_2 close to 1 it becomes very small and very large when choosing p_0 close to one. Such diffusion model can be used to reproduce the fractal shape of the interface between media starting with particles diffusing from a source to a sink into a medium (Chopard and Droz, 1998). Further, such models with diffusion front can be used to determine the percolation threshold of a system, given that clusters of media correspond to percolation sites and the probability as the parameter under consideration (Chopard and Droz, 1998). Figure 4.10 illustrates the dynamic behavior of a diffusion LGA, where particles are emitted from a source. The diffusion is shown in *a*, whereas *b* presents the corresponding moving diffusion front which is the border of the cluster of particles that are connected to the sink through neighborhood. Thus, the diffusion front denotes the percolation cluster.

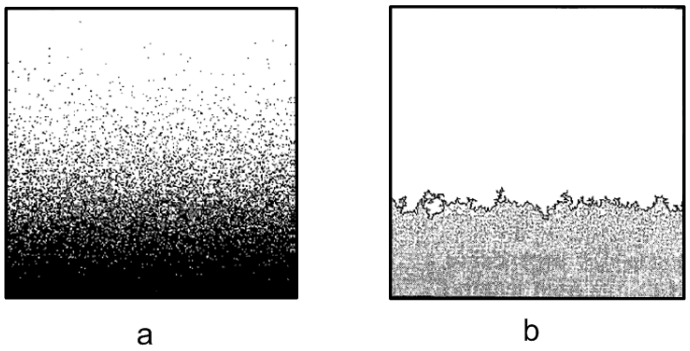

Figure 4.10: Illustration of diffusion LGA where (a) particles (black) are emitted from a source and (b) the corresponding diffusion front (from Chopard and Droz (1998)).

Based on LGA models of the transport mechanisms of fluid flow and diffusion, particles can be "equipped" with additional properties with relative ease such that aggregation, de-

position or adsorbtion results (Chopard and Droz, 1998). For example, Diffusion-limited Aggregation (DLA) can be modeled by combining a diffusion CA with "sticking" particles. This is realized by adding the notion of "rest-particles", that are particles which are immobile and occupy a site, and further adding a rule that makes a particle becoming a "rest-particle" when being in the neighborhood of another "rest-particle". Figure 4.11 illustrates a basic aggregation mechanism, where a rest particle remains and a moving particle colliding with a rest particle becomes a rest particle with a given probability (p) or else diffuses with probability $(1-p)$. The use of probability p models the avoidance of growth, where the anisotropy of the lattice is reflected in growth patterns and allows the adjustment of the speed of growth. Rotational variants are not shown in this figure.

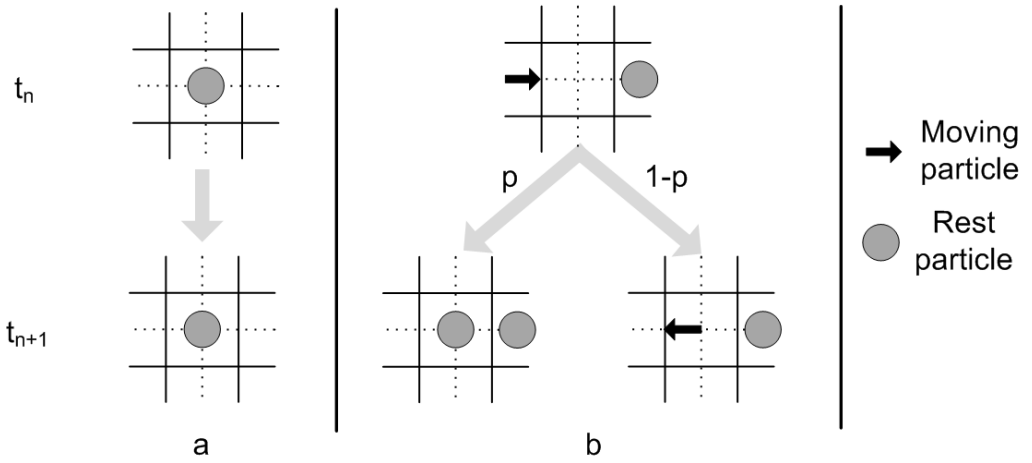

Figure 4.11: Illustration of an aggregation mechanism for LGA based on rest-particles that can be used to model diffusion-limited aggregation, deposition and adsorbtion.

Aggregation can further be conditioned by p being dependent on the local density of particles entering a site, e.g. by introduction of an aggregation threshold, thus a minimum number of particles that must be present in order to become a rest-particle. Figure 4.12 illustrates the dynamics of an exemplary LGA model for DLA, where a dendritic structure grows from a seed in the center (dark is the aggregate and gray dots represent diffusing particles). Simulations show that such models reproduce the fractal structure of aggregates and growth rates as found in many real systems (Chopard and Droz, 1998). Straightforward variations of this model have shown to reproduce the behavior of other similar processes. E.g. the usage of a nucleation surface instead of seeds reproduces the behavior (patterns and dynamics) of diffusion-limited deposition. Figure 4.12 (a) illustrates fractal growth of a DLA model as described above.

Conditioning aggregation on a surface (substrate) such that the probability of becoming a rest particle is the lower, the higher the local density of rest particles or candidate particles is, results in a model that may reproduce the mechanisms of diffusion-limited adsorbtion, where particles require some free space for becoming a rest particle (Chopard and Droz, 1998). The properties of emerging patterns under variation of diffusion and aggregation properties - e.g. the coverage and the time a stable configuration is reached - are of particular interest in investigations (Chopard and Droz, 1998). Figure 4.12 (b) shows a simulated adsorbtion surface after reaching a stable state, where the clusters of

adsorbed areas (black) are mixed with paths of with no adsorbtion that form a stable excluded area (gray).

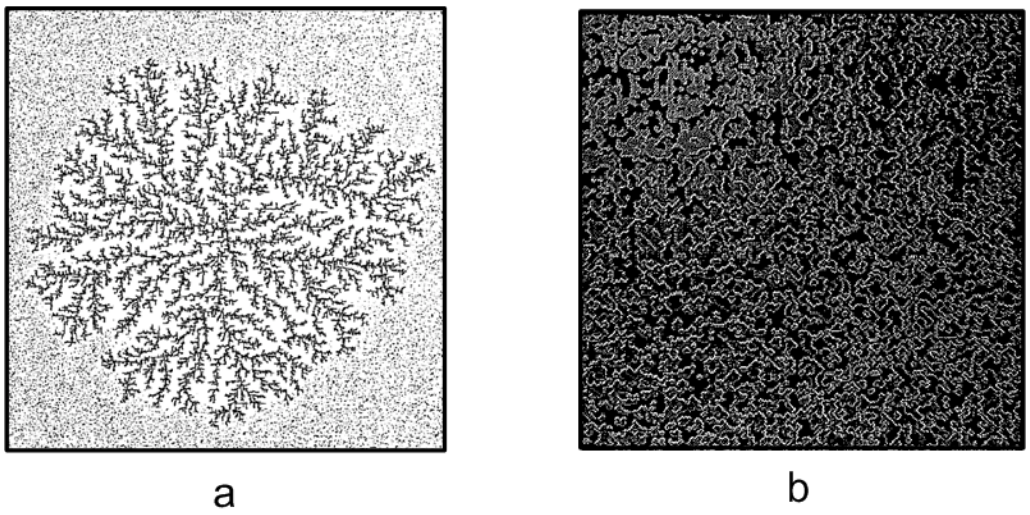

Figure 4.12: Illustration of LGA with rest particles showing aggregation behavior: (a) growth of a fractal dendritic structure (dark) following from diffusion-limited aggregation and (b) the coverage of substrate (dark) as a result of diffusion-limited deposition (from Chopard and Droz (1998)).

From the perspective of rules, models with more than one type of particles share the the property that besides diffusion, rules model an typically probabilistic aggregation behavior, where the probability of becoming a rest particle is some function of the local densities of particles of particular type, typically represented by their sum (e.g. moving, rest). Depending on the number types of particles a formulation at the base of rule tables becomes inadequate, either for reasons of space required for storage or for reasons of representation. Thus, typical representations (and implementation) include functions, such as for calculating counts of particles of a type.

Lattice-Boltzmann Method (LBM) The discrete nature of boolean CA is the source of noise in the observations, which requires averaging, e.g. spatial averaging for density observation or determination of flow. Further, the possibilities to adjust models by means of tuning parameters is often perceived as limited and for many types of systems, e.g. high Reynold number flows, the necessary number of cells is a prohibiting factor for adequate simulation studies. Against this background, the Lattice-Boltzman Method (LBM) has been developed as method for efficient simulation of LGA models and noise reduction. LBM is based on an ex-ante averaging of particle motions, where the quantity of interest - typically the number of particles moving in a direction - is not represented by an integer or boolean variable, but by means of a probability of a particle being present or the average number of particles by means of a continuous variable. The transition function calculates the corresponding probabilities/averages, which is typically defined as a derivation of from a corresponding LGA rule using averaging or factorization methods (e.g. using BKG collision term which leads to the class of Lattice-BKG models). The derivation of LBM models is specific to underlying discrete LGA and beyond the scope of this thesis. However, it must be mentioned that it introduces assumptions, such as the

neglection of many-body-correlations, that may prohibit the possibility of modeling corresponding effects, such as local fluctuations, adequately, although probabilistic modeling might introduce fluctiations to a certain degree (Chopard and Droz, 1998). Typically, LBM versions exist for a boolean LGA models for enhancing numerical efficiency and flexibility through probability tuning (Chopard and Droz, 1998). Due to the continuous nature and the possibility to tune behavior by means propabilities, LBM for fluid flow allow modeling of flows with high Reynold numbers, however some phenomena, such as specific patterns of Reaction-Diffusion systems (spirals) that are present in boolean CA cannot be reproduced, and numerical instabilities might occur (Chopard and Droz, 1998). Generally, the application of LBM in EMS is rather restricted to numerical investigations of fluid flow, such that it is not further discussed in this thesis.

Multiparticle Models Boolean LGA are limited in that the maximum number of particles moving in a direction is limited according to the exclusion principle. Multiparticle models relax the exclusion principle of strict LGA models, in that the maximum number of particles per lattice moving in one direction is not limited to one.

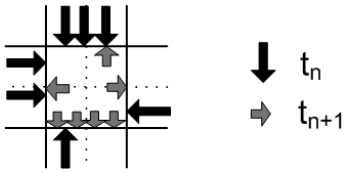

Figure 4.13: Illsutration of multiparticle CA where an arbitrary number of particles per cell and direction is allowed.

Chopard and Droz (1998) gives a corresponding diffusion model that conforms to a generalized diffusion law: At each timestep, each particle might move into one of the possible directions of movement or stay at rest, where each possibility is given a probability. If the probabilities of movement in the directions are equal, this corresponds to standard diffusion. Varying probabilities of particle movement for different directions introduces drift that allows natural modeling of advective transport. Depending on the probabilities, the size of the lattice spacing and the time step, this is shown to conform to the diffusion law with an additional advective term ($\frac{\partial \rho}{\partial t} = \vec{V}\nabla\rho + D\nabla^2\rho$, see Chopard and Droz (1998)).

A multiparticle fluid flow model is more complicated. Chopard and Droz (1998) presents a model where at each timestep each particle is preliminarily distributed with a given probability p_i to the possible i directions of motion. p_i is derived from the geometry of the lattice and Navier-Stokes equations (Chopard and Droz (1998)). Afterwards the conservation of momentum is tested by summing up particles in the directions. If momentum is not conserved a particle randomly redistributed and momentum conservation is tested. This procedure is iterated until local momentum conservation is satisfied.

Multiparticle models naturally provide possibilities to extend boolean CA, in particular when particles are further equipped with behavior (see below). Discreteness provides numerical stability, however, the simulation procedure is more complex than boolean CA and LBM since it contains loops over particles within cells and further iterations (multiparticle fluid) with corresponding random number generation. However, great numbers of particles per cell (> 40) allow the consideration of bulks of particles with the same behavior, e.g. the probabilistic sampling of the number of particles moving in a particular direction

instead of considering each particle separately, such that random number generation is not necessary for each particle (Chopard and Droz, 1998).

However, the formulation of rules is not possible at the base of lookup tables and it typically involves functions (e.g. summation) considering a possibly infinite number of particles, which requires algebraic calculations at the time of simulation. Thus, there is a considerable computational effort related to multiparticle LGA in particular when compared to boolean LGA, for which computations may be highly optimized (see below).

Reactive LGA It is a common approach to combine explicit movement of particles (diffusion and flow) with reactive behavior. In particular the combination of diffusion with reaction appears to be a mechanism of some generality, in particular with respect to the formation of spatial patterns.

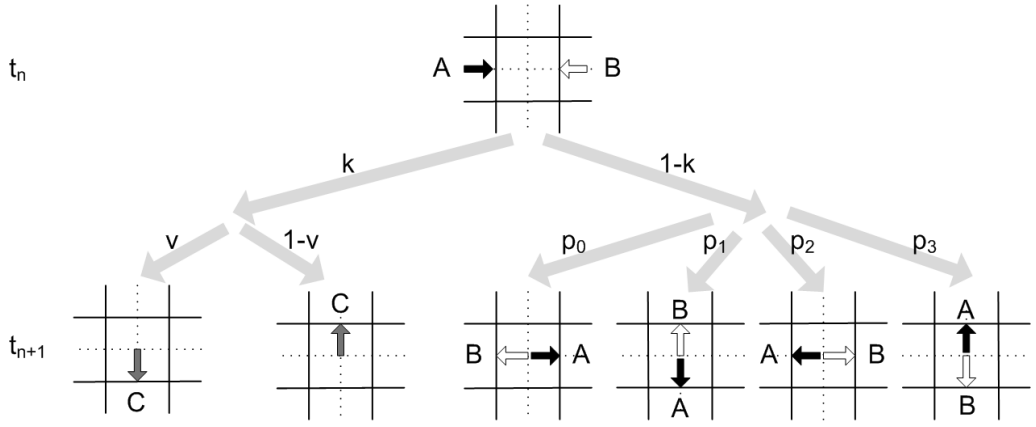

Figure 4.14: An boolean LGA rule for a Reaction-Diffusion process.

Figure 4.14 illustrates a simple CA model for Reaction-Diffusion, where A and B-particles react to C, with a reaction constant K, while diffusing. The model is probabilistic in that there is a probability of reaction (k), which is associated with reaction constant K and a probabilistic choice of direction after collision (p_i and v, from Chopard and Droz (1998). Typical variation of this model are introduced by changing probabilities and by adding rules for reactions involving varying numbers of particles (Chopard and Droz, 1998). Each type of particle might be considered for diffusion independently of the other types (multiple lattice model), such that the exclusion principle holds only for each species independently, in contrast to single lattice models (e.g. Kapral et al. (1991); Dab et al. (1991)). Using different probabilities of change of direction or updating schemes for the diffusion of different species, different diffusion constants can be modeled (Kapral et al., 1991). However, due to the limited number of lattices and the exclusion principle, there are limitations in modeling stoichiometric constraints (proportions of species in reaction) and high-concentrations of species, which may be overcome by multiparticle LGA (see below).

The multiparticle model allows intuitive modeling of reactions of the form $mA + nB \rightarrow C$, with multiple particles of several types per site and direction, such that stoichiometric constraints (here: the ratio $\frac{m}{n}$) can be naturally considered. Particle transport is given by an respective diffusion model (see above), where each type of particles might proceed with a different speed, adjusted by different probabilities or different update frequencies of

diffusion rules. Given the prerequisites of a reaction are given at a site (i.e. the presence of m particles of species A and n particles of species B) a reaction might take place with a probability k that represents the reaction constant.

Chopard and Droz (1998) provides a rule that models reactions within multiparticle models, where priorities of different possible reactions, conformance to stoichiometric constraints and availability of particles must be considered, typically by probabilistic application of rules, with the reaction constant modeled by a reaction probability (Chopard and Droz, 1998). However, the description of multiparticle rules requires separate consideration of particles at all sites and algebraic description of probabilistic rules, where a specific number of particles reacts with a specific number of particles to produce a specific number of particles.

Chopard and Droz (1998) provides a multiparticle CA that has shown to conform to the corresponding rate equations: For each cell first a reaction step is considered, before the resulting particles are distributed according to diffusion. The reaction step calculates the number of possible combinations of particles at a cell that allows a reaction (i.e. $N = \binom{l_A}{m}\binom{l_B}{n}$, where l is the number of present particles of a species and m and n give the necessary number of corresponding for a reaction). Then for maximally N times the reaction is executed with probability k, where for each actual execution reaction the balance of the corresponding particles must be updated. If there are not enough particles left for reaction, the execution of reactions is stopped. Thus, the model includes loops over particles, per-particle probability consideration and bookkeeping of particle numbers in order to preserve quantities. Thus, the formulation of rules in terms of tables is not feasible and instead relatively advanced computations have to be specified, generally considering arbitrary amounts of particles.

A prototypical example of pattern formation in reactive LGA is the implementation of supersaturation mechanism for the investigation of Liesegang pattern formation. Liesegang pattern formation can be modeled by extending the simple Reaction-Diffusion LGA with the process of precipitation, where a diffusing C particle may be transformed into a resting D particle. Precipitation is implemented according to the principles of supersaturation theory: if the local density of a solute reaches a threshold, precipitation occurs (nucleation threshold P_{nucl}). Further, C particles aggregate around D particles depending on density threshold (aggregation threshold P_{agg}) and may eventually transform into D, depending on a threshold (transformation threshold P_{trans}). These different thresholds are the main control parameters of the model and can be used to quantify the existing qualitative models of solidification that relate supersaturation and growth behavior Chopard and Droz (1998). Figure 4.15 illustrates a corresponding simulation showing the formation of Liesegang patterns.

A further prototypical mechanisms that leads to the formation of spatial patterns is that of the formation of Turing patterns at the base of inhibitor-activator interaction. Here two chemical species diffuse with different speed and react, such that the product of the reaction is one of the inputs (autocatalytic reaction). A common characteristic of such systems is critical behavior around bifurcation points that are characterized by a specific ratio of diffusion constants. A prototypical example for the investigation of associated phase transitions (bifurcations) and the formation of spatial patterns is the Schnakenberg model. The Schnakenberg model is an inhibitor-activator model that is widely used for the investigation of Turing patterns under particular consideration of fluctuations, which are

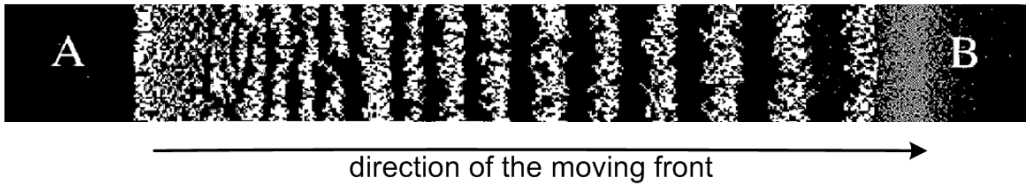

Figure 4.15: Example of a LGA Reaction-Diffusion CA model showing the formation of Liesegang patterns, where stable white bands are formed by the precipitate that follows from A diffusing from left and reacting in within a uniform distribution of B (from Chopard and Droz (1998)).

not included in mean-field density-based reaction equations[15]. The Schnakenberg model is a autocatalytic reaction described by $A \xrightarrow{k_1} X$, $X \xrightarrow{k_2} \emptyset$, $2X + Y \xrightarrow{k_3} 3X$, $B \xrightarrow{k_4} Y$. A prerequisite for the formation of Turing patterns is a difference in diffusion coefficients, thus the velocity of the two reactants, where the inhibitor (i.e. Y) is faster than the activator (i.e. X).

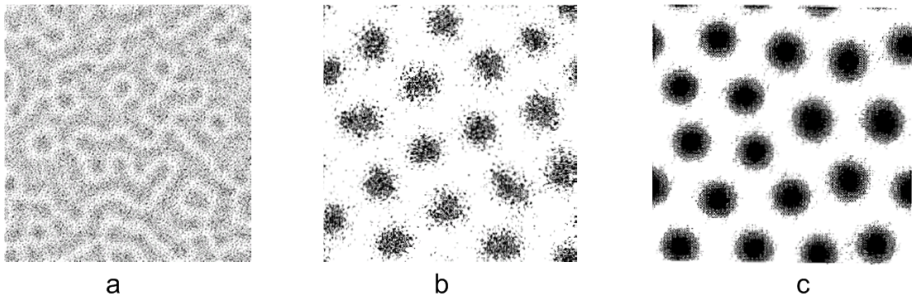

Figure 4.16: Observed pattern formation in the Schnakenberg model (from Chopard and Droz (1998)).

Figure 4.16 presents a snapshots of simulations of the Schnakenberg CA model. Figure 4.16 (a) shows a snapshot of a simulation at a ratio of diffusion coefficients ($d = 30$) that appears to be near the parameter where the Turing instability (bifurcation) occurs and where small fluctuations may lead to pattern formation instead a homogeneous steady state. Figure 4.16 (b) shows pattern formation for a CA model and (c) a numerical simulation of rate equations of the corresponding Schnakenberg model (from Chopard and Droz (1998)). Please note, that in contrast to the equation-based model, which is stationary after a certain time, the CA model is subject to permanent local fluctuations. In systems with Turing instability such fluctuations may permanently cause instability and even the reorganization of patterns, as illustrated in Figure 4.17, at a CA simulation for the Maginu model[16] (see Dab et al. (1991)).

The above prototypical examples of LGA and LGA-based models exemplify basic mechanisms that have been considered in many variants, specializations, generalizations and combinations. Indeed there are three-dimensional variants of the basic transport mech-

[15] A variety of similar models exist (see Schnakenberg (1979)) the most famous are the *Brusselator* and the *Selkov* model.

[16] The Maginu model is a simple Turing model $\frac{\partial x}{\partial t} = \Psi(x,y) + D_x\nabla^2 x$, $\frac{\partial y}{\partial t} = \Phi(x,y) + D_y\nabla^2 y$, with $\Psi(x,y) = x + \frac{x^3}{3}$ and $\Phi(x,y) = x + \frac{x-ky}{c}$

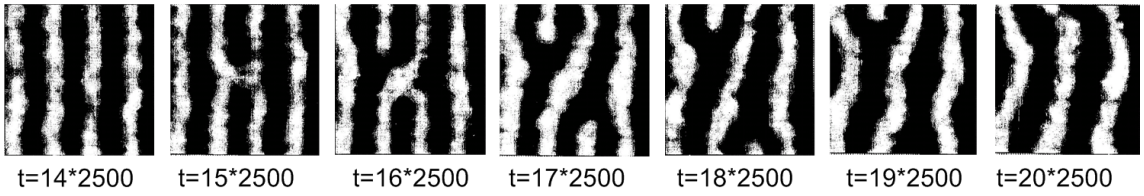

Figure 4.17: Reorganization of a pattern (from 4 stripes to 3 stripes) due to Turing instability and local fluctuation simulated by boolean coupled lattice diffusion-reaction LGA CA of the Maginu model (from Karapiperis (1997); Dab et al. (1991)).

anisms of flow and diffusion (Chopard and Droz, 1998). The combination of transport mechanisms and particle-based interaction of matter is common not only for diffusion processes (e.g. reaction-diffusion, DLA), but also interaction of liquids (multiphase flows) can be modeled by means of combining LGA fluid flow with interaction. This encompasses models (boolean LGA and LBM) for miscible, immiscible and reactive fluids, where typically fluctuations have considerable influence on behavior and interfaces may be of complex shape (fractal).

Ising spin models

Like the HPP LGA, the Ising spin model is one which has been developed first without reference to CA, but which has later been perceived as CA. The Ising spin model is originally concerned with modeling magnetization which is perceived as a collective behavior occurring depending on a critical temperature T_c (no magnetization for temperatures above T_c and magnetization below T_c).

Generally, the term *Ising model* refers to models that are made up of spatially arranged elements that have a spin that is either "up" or "down". Spins interact due to a coupling energy $\pm J$ that occurs when neighboring spins have opposite direction. The alignment of large clusters of spins is the magnetization observed at the macro-scale. The total energy of Ising model is constant and given in the initial configuration. Figure 4.18 (a) illustrates a configuration of spins and corresponding energy.

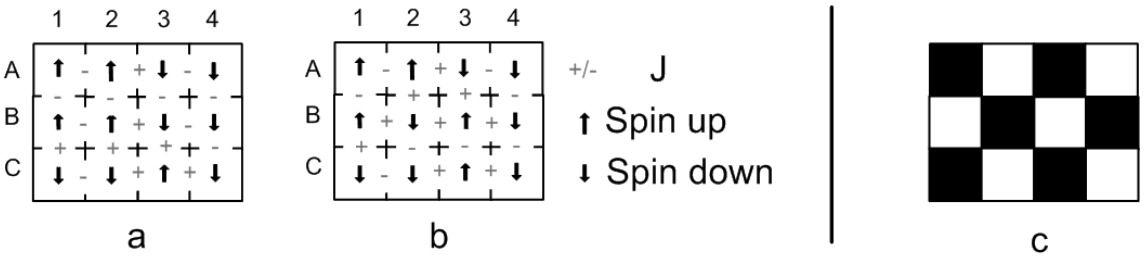

Figure 4.18: An exemplary configuration of an Q2R Ising CA (a), a configuration resulting from synchronous update of cells 2B and 3B with a different energy level and chessboard asynchronous update scheme for energy conservation (c).

The most famous and exemplary Ising spin CA model is the Q2R model (see Chopard and Droz (1998); Vichniac (1984)), which is built on the premise of local energy conser-

vation. Aligned spins contribute energy $-J$ opposing spins add energy J. Spins flip, if the local energy is preserved, thus the number of neighbors with spins "up" and "down" are equal. Figure 4.18 (b) illustrates how synchronous update changes the energy level if neighboring spins are flipped synchronuously (i.e. cells B2 and B3). For this, asynchronous update is required, e.g. the update of one cell per timestep. A computationally more efficient variant is the "chessboard update" meaning that there is an alternating update of non-neighboring cells (e.g. for two sublattices that correspond to white and black squares of the chess board in Figure 4.18 (c)).

Formally, Q2R-rule can be described by

$$s_{ij}(t+1) \begin{cases} 1 - s_{ij}(t) & \text{if } b_{ij} = 1 \text{ and } s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} = 2 \\ s_{ij} & \text{otherwise} \end{cases}$$

and

$$b_{ij}(t+1) = 1 - b_{ij}(t)$$

where s_{ij} denotes the spin at coordinate (i, j) and b indicates if update it due.

Figure 4.19 illustrates a typical experiment within a series of experiments investigating the correspondence of the global energy level (T), that is encoded in the initial state - (s_{ij}, with values 1 and -1) - (a), and the emerging pattern of magnetization (b,c,d, from Chopard and Droz (1998)) . Initial configurations are typically generated randomly (Monte Carlo) based on a given probability (p) for a spin being "up". Varying p within experiment series allows the determination of the critical Temperature T_c and a characterization of behavior at values near T_c.

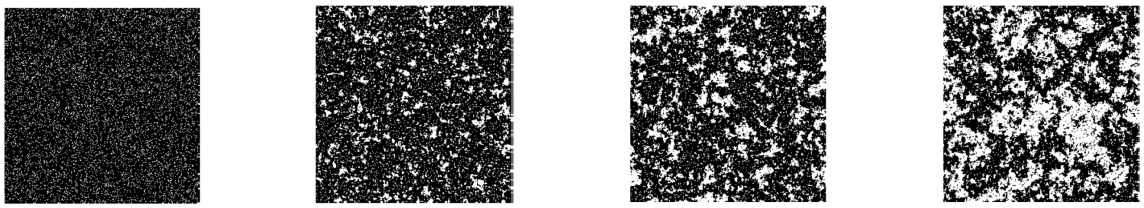

Figure 4.19: Evolution of Q2R CA at different times: intial state encoding the energy level (a), transient states (b,c) and stable final configuration (d, from Chopard and Droz (1998)).

Q2R is reversible and therefore shows interesting behavior in that it never reaches homogeneous state at low energy level, since there is always a fraction of spins up and spins down. This corresponds to observed non-zero magnetization, but also shows conservation of information, in that the Ising model "remembers" the initial state. Although the Ising CA is a crude abstraction of real workings at the micro-scale - which are dominated by quantum effects -, it appears to incorporate some basic aspects of its workings. In particular, it is an example of how irreversible macro-scale behavior can be the result of reversible micro-scale behavior (Chopard and Droz, 1998; Vichniac, 1984).

The Ising spin interaction can be combined with LGA-based particle transport where particles have a spin and interact according to spins. This is an example how particles can be equipped with additional properties, such that interesting behavior emerges. Chopard

and Droz (1998) present a combination of FHP fluid flow model and Ising model, where particles move, but preserve mass, momentum and spin. Thus a shuffling of directions at collisions occurs according to FHP but spin is preserved, thus two types of particles are distinguished. To the movement a local interaction of spins at a cells and between neighboring cells is added, such that spins flip, if the local energy (E_i) is lowered by that, otherwise they flip with a probability W, which is a function of the local spin configuration (see Chopard and Droz (1998) for details). The local energy is calculated as the sum of coupling energy at the cell and between neighboring cells. Update is asynchronous with three sublattices, such that neighboring spins are not updated at the same time. The FHP rule and the spin update are applied in an alternating way and the variation of relative frequency of the rule allows tuning of the model, thus weighing the effects of flow and spin interaction.

With spin dynamics, the fluid acts like an Ising system in that the fluids (spins) spatially organize in an emergent way depending on a critical temperature (T_c). Figure 4.20 (a) illustrates the emergence of an organized pattern in which two fluids segregate through the formation of domains below a critical temperature. Figure 4.20 (b) illustrates the behavior of a Ising model of two immiscible fluids model that models the Raleigh-Taylor instability, where a dense fluid lies on top of a less dense fluid (left). Local fluctuations destabilize the configuration and lead to the emergence of mushroom-like patterns.

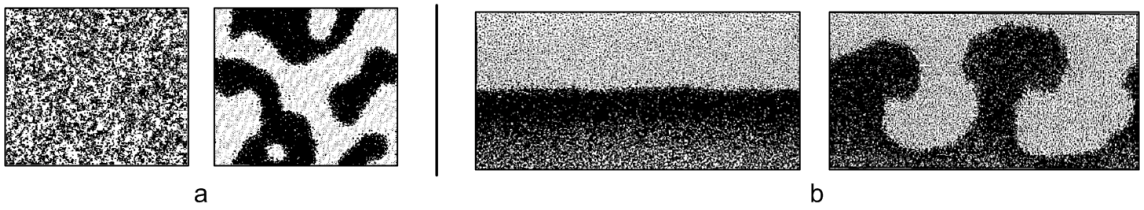

Figure 4.20: Ising fluid examples: Emergent organization of two fluids (a) from an unordered initial state (left) to organized (right) and (b) behavior of two imminscible fluids (Raleigh-Taylor instability), where a local fluctuation (left) leads to a mushroom-like pattern (right, from Chopard and Droz (1998)).

Excitable Media

The mechanism underlying models of excitable media is an example of a simple local mechanism producing complex spatial patterns. The study of excitable medium (e.g. Belousov-Zhabotinsky) is associated with CA models with cyclic states (note: not cyclic boundary conditions). Here the state of a cell is perceived to go through a number of phases, which are cyclic. Transition between states is typically adjusted by means of thresholds and randomized through probabilistic modeling (Monte Carlo). A prototypical example is the Greenberg-Hastings model of excitable medium, where a cell's state is either resting, excited or refractory. In the resting state a cell is stable, through an perturbation the cell can get into excited state where it influences neighbors, before it gets into the refractory state, where it does not influence neighbors and is not excitable (Chopard and Droz, 1998). A cell becomes excited when the number of excited cells in the neighborhood exceeds a threshold (excitation threshold th_e), thus the Greenberg-Hastings model is a counting CA with threshold dynamics. The model is very sensitive to excitation threshold and the time spent in excited t_e and refractory state t_r (Chopard

and Droz, 1998). Parameters of Greenberg-Hastings are typically the times spent in the different states and the excitation threshold (Zhao et al., 2007)).

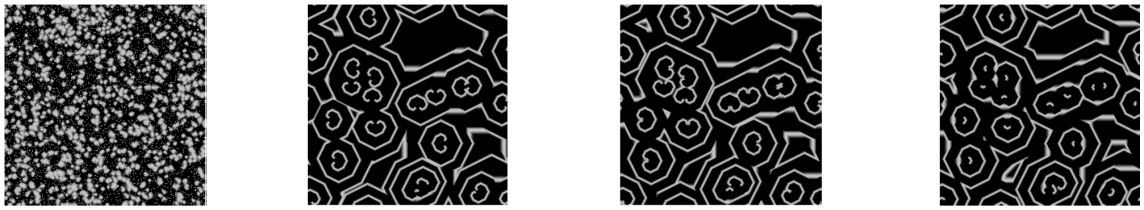

Figure 4.21: Evolution of a Greenberg-Hastings CA with $th_e = 3$, $t_e = 4$ and $t_r = 5$: After a transient phase (a), the system shows pairs of counter-rotating spiral waves, that evolve to new similar patterns when extremities meet (from Chopard and Droz (1998)).

In general a number of CA models with cyclic state have been shown to be able to reproduce complex patters such as Turing patterns or spirals, in which local fluctuations play a major role (see Chopard and Droz (1998); Toffoli and Margolus (1986); Vichniac (1984)). Figure 4.21 illustrates the generation of patterns at the example of a Greenberg-Hastings model.

Voting Rules

Voting rules - a specialization of counting rules - are rules where the maximum count of a state within the local configuration prescribes the next state, possibly based on a threshold. Vichniac (1984) shows that voting rules generally give rise to critical behavior with processes of nucleation and or the relaxation to a percolating state is a typical. Such rules are sensible to the initial configuration and have been shown to generally allow investigations of critical behavior (e.g. determination of critical values of parameters, e.g. percolation threshold) by means of encoding the value of corresponding parameters within the initial state and subsequent simulation. Relaxation to percolating behavior proceeds in that clusters of one state grow at the expense of the other finally forming a percolating cluster, but clusters of all states remain present. Nucleation occurs through the formation of compact clusters from local high-density fluctuations that subsequently grow, leading to isolated clusters or homogeneous state. A typical application of models with nucleation (formation and growth of clusters) is the investigation of phase transitions, in particular the identification of critical parameter values and the behavior. Voting rules provide simple mechanisms that produce spatial behavior and patterns observed in real systems, the occurence of which is associated with behavior at the critical values.

The probably most famous example of self-organization of voting rules is the simple "twisted majority" rule (or annealing rule). It refers to a boolean CA with von Neumann neighborhood that assigns a 1, if the count in the local configuration is 4 or $\geq$ 6, 0, else. This rule encourages a reshuffling of votes with marginal majority and leads to the annealing of domains in the long run. It has been shown that it reproduces the motion and curvature of the interface between two phases in accordance to existing theory of fluids, where the inherent surface tension is emergent (Chopard and Droz, 1998).

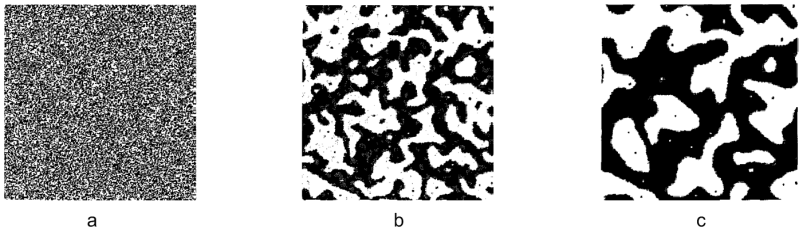

Figure 4.22: Evolution of CA with "annealing rule": From a random initial condition (a) clusters are formed and concavities filled (b,c) according to an inherent surface tension (from Chopard and Droz (1998)).

Percolation and SOC in CA

A variant of CA models are percolation models, which provide a template for a number of related types of models. Percolation models serve the purpose of investigating properties of percolation, e.g. finding the percolation threshold, thus identify the critical value for p above which percolation occurs, thus there is a connected cluster spanning the the complete area. Figure 4.23 illustrates a site percolation model with $p = 0.4$ below the critical percolation threshold, thus no percolation, and $p = 6$ above the percolation threshold with percolation. The typical application of the idea of percolation in the framework of CA is to cast percolation into a probabilistic model of movement of entities (e.g. particles). Cells are perceived as sites between which the entity may move constrained by the probability p that either denotes a probability of a neighboring cell being connected (bond percolation model) or a neighboring cell being occupied, such that movement is (not) possible. A typical template for a corresponding CA is that for all cells with an occupying entitity, all cells or some cells from the neighborhood are chosen randomly and an entity is placed in it with probability p so that it is added to the cluster of occupied sites.

Figure 4.23: Site percolation model with $p = 0.4$ (a), $p = 0.6$ (b) and an epidemic forest fire model ($p = 0.6$) at time 70 and with isotropic rule (c) and anisotropic (d) with $p = 0.6$ for horizontal and $p = 0.65$ ($p = 0.45$) for northward (southward) movement (from Boccara (2004)).

Figure 4.23 (c,d) illustrates a corresponding simple epidemic model of forest fire that adds a cyclic state ("tree present": gray, "tree burning": white, "empty". black) to percolation-based movement of fire. An isotropic movement follows from all probabilities being equal (c), anisotropy - e.g. considering wind - can be introduced easily by means of different probabilities for the different directions (d). Like the simple site-percolation model, the isotropic model has critical behavior with critical parameter value $p = 0.5$.

Anistropic versions have different critical parameter values (Boccara, 2004).

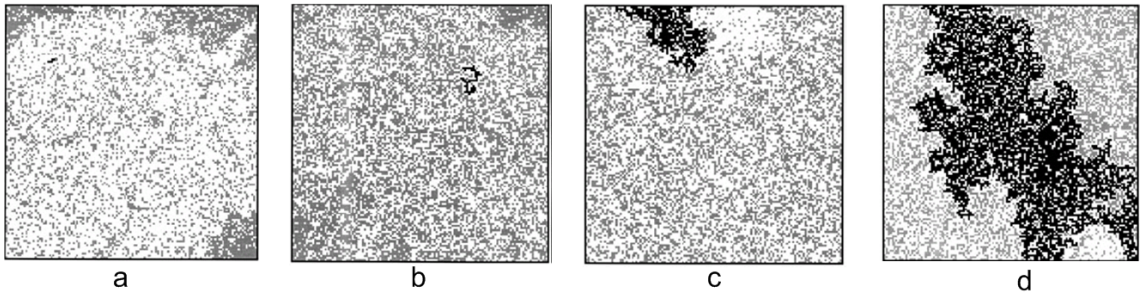

Figure 4.24: Forest fire model with critical percolation and SOC behavior (from Turcotte (1999)).

Another forest contains a percolation model for tree growth and combines this with a probabilistic forest fire model exhibiting SOC behavior. Figure 4.24 illustrates the behavior of this forest fire model, consisting of a site percolation model for tree growth and a epidemic model for fire forest fire: (i) a burning tree becomes an empty site, (ii) a tree burns if at least one neighbor is burning, (iii) at an empty cell a tree grows with probability p and (iv) a tree without burning neighbor becomes burning with probability f. Without fire, this model is a simple percolation model for forest growth with a critical parameter value for p where the size of forest clusters diverges. In total, the model shows self-organized criticality in that fire sizes follow a power-law distribution (Drossel and Schwabl, 1992). Figure 4.24 illustrates the occurrence of fires of all sizes in this forest fire model (from Turcotte (1999))

The framework of CA appears to be particularly amenable for modeling systems with SOC, since the mechanism underlying SOC naturally aligns with CA: An external source constantly adds some quantity to the system which accumulates in a spatially heterogeneous (initial condition) way (in the cells). If the quantity reaches a particular local threshold it is redistributed among the neighbors, which in turn might reach their threshold and so on. An important ingredient of SOC is that the addition of quantity is considerably slower that its redistribution. Due to the generality of the mechanism, a variety of models have been developed and investigated that encompass this mechanism of SOC in order to find an explaining mechanism for a variety of observations that show power-law distribution and fractal shapes (e.g. forest fires, earthquakes, landslides).

4.3.4 Method and Pragmatics

Pragmatic aspects of modeling with CA at the micro-level are well described. In particular the comprehensive works Wolfram (2002), Chopard and Droz (1998) and Toffoli and Margolus (1986) devote large parts to the description of pragmatic aspects of CA-based modeling. The authors emphasize that the CA modeling paradigm is perceived to align with a rather unconventional explorative approach to investigation: Toffoli and Margolus (1986) rather emphasizes that the developments in the field of digital computing facilitate an explorative approach of investigating "physically minded" CA models as opposed to rather deductive approach to mean-field type modeling with differential equations. Associated is a level of abstraction typically at the level of atoms or molecules - corresponding to the abstraction of "particle" -, where two kinds of simplifactions are fundamental: first

quantum effects are not considered fully (superposition and entanglement) and, second, the number of particles is limited to a number such that it is perceived to mimick behavior such that expected macro-scale properties emerge, not the real quantity of corresponding molecules, which is some orders higher (e.g. LGA and Ising models)[17]. Within this approach to CA-based modeling, investigations typically aim at reproduction of systems behaviors comparable to what is reproduced by traditional approaches, such that it can be used for prediction in practical applications.

In an attempt to promote "a new kind of science", Wolfram (2002) even postulates that the paradigm of CA might represent the most fundamental mechanism onto which all phenomenology is built upon, based on the assumption that anything in the physical world is computation or is equivalent to computation. CA appeared as the framework with a high-level of simplicity that generally gives rise to arbitrarily complex computation (universality). Besides the general mostly analytical considerations related to computation and physics (see Chapter 4.2), the investigation of CA for the purpose of investigating specific phenomena is necessarily built upon simulation since interesting properties of CA are typically not decidable, thus they have to be evaluated empirically ("computational irreducibility"). However, a great deal of research is related to find patterns in rule sets of CA at the base of relating observed CA behavior with characteristics of rules, such that general mechanisms or at least intuitions can be supported.

From a pragmatic point of view, the studies that correspond to the approaches follow different typical procedural templates. The investigation of general mechanisms of CA typically starts with the identification of a class of CA that is to be investigated. Classes are typically designed such that they are most simple and allow for exhaustive simulation. Besides elegance, simplicity should enable at least some kind of analytical treatment for the identifications of algebraic properties, although it has been shown that most properties are not decidable even for simple CA. This kind of investigation started with the investigation of elementary CA and has been extended to other classes, e.g. voting rules (Vichniac) within 2-D boolean CA. Experiment series that exhaustively traverse the rule space of the class produce trajectories that are subsequently classified according to the property under consideration (e.g. Dynamical System class, generated patterns). Classification is typically based on statistical evaluation of outputs and visual inspection. Correlations between measurements on rules and the classes define an ordering of rules, that might be used to guide the conceptualization of further CA as an aid to intuition or more efficient (non-exhaustive) traversal of e rule spaces. The correspondence to real systems - thus validity - is evaluated empirically at the base of rather qualitative similarities for which quantifications (e.g cycles, proportions of cells, growth rates) may be defined.

The more physically-minded approach is rather different from a pragmatic point of view: Starting from a physical intuition (e.g. particle movement) and possibly according to general laws (e.g. conservation laws), rules at the micro-level are designed such that they conform to the constraints by construction. Analytical investigations may prove conformance or deviance of resulting macro-scale properties that are derived from the same general laws (e.g. Navier-Stokes, see Chopard and Droz (1998) and Boccara (2004) for details). Given this kind of analytical validation[18] CA models can be used to overcome limitations of the conventional equation-based approach. From a rather technical per-

[17] For this reason CA-based modeling is rather located at the "meso-scale" between models at the molecular level and macro-level.

[18] It is assumed that the stated macro-scale properties are true.

spective the most important limitations of equation-based approaches appear to be the performance of simulators, the neglection of small scale fluctuations and the complicatedness of dealing with fractal interfaces, which appear common in non-linear systems. Further, it appears that CA provide relatively good means to investigate phase transitions in open system, where these not only provide means to empirically identify macro-scale properties analogously to the closed system case, such as critical parameter values, but they also provide explicit behavior near critical points, such as the formation and static properties of spatial patterns (e.g. percolation, cluster size and shape), that appear as changes of stability (e.g. eigenvalues) within the conventional framework.

The above presented typical fields of investigation and associated models illustrate that these approaches are the extremes of a continuum of issues and related practices of investigation. The field of investigation of diffusion-reaction systems is exemplary in that there is a sound theoretical basis at least with respect to transport and reaction mechanisms independently and systems are closed, but when these processes are intermingled and systems open, thus do not reach equilibrium, not only numerical problems may exist, but also the theoretical and methodological underpinning may be fragmentary. This means that the theoretical basis is not sufficient to fully derive models, thus solve a forward problem, but there are CA that have shown to provide some mechanisms that - at least qualitatively - reproduce phenomena (e.g. pattern formation) that are subject to consideration. However, the application to specific issues requires adaption of general mechanisms, which typically comes along with relaxation of the simplicity constraint, while the fundamental mechanism are tried to be preserved by construction. This, and the relative ease with which processes can be combined, e.g. by equipping particles with interactive behavior, within CA-based models appears to be a major pragmatic aspect of the use of CA.

Achieving quantitative agreement with observations with real systems typically involves the extension of states, such that it corresponds to perceptions of real systems, which is typically not binary. Further, required adjustments relate to the rate of evolution (e.g. velocities, reaction rates) and the removal of artifacts that result from discretization and simplicity (e.g. unwanted conservations, and anisotropy). Adaption and combination of processes also leads to the introduction of additional states, compared to general models, that typically use binary state. Further adaptions are the introduction of an additional lattice dimension and continuous state variables (when probabilities are involved). Further, inhomogeneities with respect to time and space are being introduced in that different rules are applied at different times or different locations of the CA (e.g. different diffusion speed of different species or chessboard update).

Probabilistic modeling is a widely used approach for this since it often allows the preservation of the explicit formulation of main mechanisms by making randomness entering externally. In *probabilistic CA*, there are several possible configurations that may follow from a given given configuration, but a probability is given for any possible transition (e.g. $\delta^{loc} : Q^m \times Q \rightarrow [0;1]$, defines a probability of entering a specific state ($q \in Q$) given a local configuration ($c^{loc} \in Q^m$, where the sum of probabilities equals 1). The introduction of probabilities allows the continuous adjustment of discrete-state CA models as do parameters in macro-scale models, given the number of probabilistic events allows for statistically meaningful aggregation (e.g. diffusion constant and reaction constants). By this, the adjustment of threshold dynamics is possible by means of probabilistic conditioning, even with few states and small neighborhoods and probabilistic modeling helps to overcome anisotropies of the CA lattice geometry (e.g. aggregation) or to introduce wanted anisotropy (e.g advection), while preserving the explicit representation of the mechanism

under consideration. Further, probabilistic CA may be used as a substitute for generally non-deterministic CA.

Although the synchronicity of update of cells is commonly perceived as a defining feature of CA, and corresponds to the physical intuition that events are typically not completely ordered, in practice, explicitly asynchronous update is often introduced (e.g. Q2R). The reasons for the introduction of asynchronicity range from fundamental, e.g. where full synchronicity is not perceived to be a real property of the system but rather an artifact of the chosen level of abstraction (e.g. Ising system), to practical, e.g. where the frequency of updates adjusts the speed of evolution. To be distinguished are regular forms of asynchronicity, where the update pattern is described deterministically as a deterministic function of time or space, from non-deterministic forms of asynchronicity, where it is non-deterministically chosen for each cell if an update is applied. In case of non-determinacy, the update scheme is typically modeled probabilistically such that a probability is given for a cell to be updated either by explicitly assigning a probability for a transition with no state change or by probabilistically choosing cells for transition. Regular asynchronous update schemes (e.g. "chessboard update", cyclic application of rules with given frequencies) might be simulated by means of synchronous CA with additional states that indicate update status[19].

Spatial inhomogeneities are those where different transition functions are specified for cells at different positions within the lattice. Boundary cells typically introduce spatial inhomogeneity, since special update rules may be given for them. Other applications of spatial inhomogeneity may reflect heterogeneous spatial configurations with different properties. Please note the distinction between the conceptualization and the implementation of properties of probabilism, asynchronicity and inhomogeneity: From a purely formal perspective, asynchronous update can be formulated as a probabilistic rule, or spatial inhomogeneity, which again might be modeled at the base of additional state variables and according rules (e.g. Q2R Ising model). Thus, this distinction rather reflects the conceptualization than the implementation and simulation of models.

In general, the variety of types of CA models shows that CA is not so much perceived as a rigid mathematical framework in the field of modeling physical processes, but rather as a philosophy, that is put into opposition of traditional continuum approaches with differential equations (see Chopard and Droz (1998); Toffoli (1984)). The aim for simplicity is major goal, where it is hoped that CA may provide mechanisms that will show generality across disciplinary borders (Vichniac, 1984). The deviation from the classic model of CA-based modeling appears to be the rule rather than an exception, the more models are to be designed in order to replicate quantitative aspects of real systems.

4.3.5 Tools and Languages

General Aspects of the Simulation of CA

The development and application of CA parallels the developments in digital computing. Generally, simulation-based studies are limited by available computational resources, in particular probabilistic modeling requires a statistically sound amount of samples, thus simulation experiments, and micro-scale modeling basically requires an amount of entities and events (e.g. particle movements and collisions) that allows statistical aggregation

[19] E.g. The Q2R CA simulating an Ising system with "chessboard update" (see Chapter 4.19) is implemented with two additional states indicating the phase of a cell and corresponding rule

such that it can be related to continuous macro-scale theory and observations. Classic CA appear to be particularly well aligned with digital computing, which is associated with temporal and spatial homogeneity, discreteness, locality of interaction and simplicity.

In general, the temporal and spatial homogeneity allows classic CA to be simulated by an inherently simple simulation procedure, which - in the serial case - is structured in nested loops, where, the outer loop traverses all timesteps and nested inner loops for each dimension of the CA (see Algorithm 1). For each cell (location), the local configuration is retrieved by the application of the localization function and the local transition function is executed. The local configuration is typically directly retrieved from memory in form of direct access to the state matrix via index or by reference into memory and not calculated by means of a function. The synchronicity of updates requires the result of the local transition function to be stored in an independent data structure (*newState*) that is copied to the data structure holding the state after all applications of the local transition function. In practice, typically references to the data structures are simply swapped, thus there is no actual copying of the contents of the matrices (Worsch, 1999).

Algorithm 1 Simulation procedure for classic 2-d Cellular Automata (pseudocode)

```
int [maxX] [maxY] state          ▷ State matrix holding values of global configuration
int [maxX] [maxY] newState                    ▷ Matrix holding intermediate values
for t++ < maxT do                                  ▷ Loop through all time steps
    for i++ < maxX do                               ▷ Loop through state matrix
        for j++ < maxY do
            confLoc ← locFunction(state, i, j)                    ▷ Retrieve c^loc
            stateNew [i][j] ← deltaLoc(confLoc)                     ▷ Apply δ^loc
        end for
    end for
    state ← stateNew                  ▷ Assign new values to the state matrix
end for
```

If the local transition function is formulated as a rule table, the execution of the transition function corresponds to the identification of the rule within a lookup table and subsequent retrieval of the resulting value, with no further calculations. This requires a limited set of discrete states and discrete neighborhood. This is efficient, when the complete rule table fits into the fast access memory of the processor. Generally, the size of the rule table is the number of possible local configurations, which is $|\tilde{Q}|^{|N|}$, where $|\tilde{Q}|$ is the number of states and $|N|$ the number of neighbors. Further, a small number of states may be considered for optimization: if the number of bits needed for representing the state of a cell b is smaller than the word length of the machine w, several states might be represented by one machine word, such that the number of memory accesses might be reduced. Indeed this requires a corresponding specification of the local transition function at a rather low level (Worsch, 1999). Further, b machine words can be used to store w states where each state is identified by its bit position within the machine words.

In general, due to locality of interaction and synchronous update, the classic CA approach naturally lends itself to parallelization using the technique of *domain decomposition*, which can be used to exploit parallel hardware with several processors. There exist a variety of basic approaches to design parallel computation, from heterogeneous networks of connected workstations, homogeneous bus-connected distributed or shared memory settings to relatively integrated multi-core processors. Computational performance - thus

optimizations - are specific to the hardware and the characteristics of the model under consideration and a comprehensive overview is beyond the scope of this thesis.

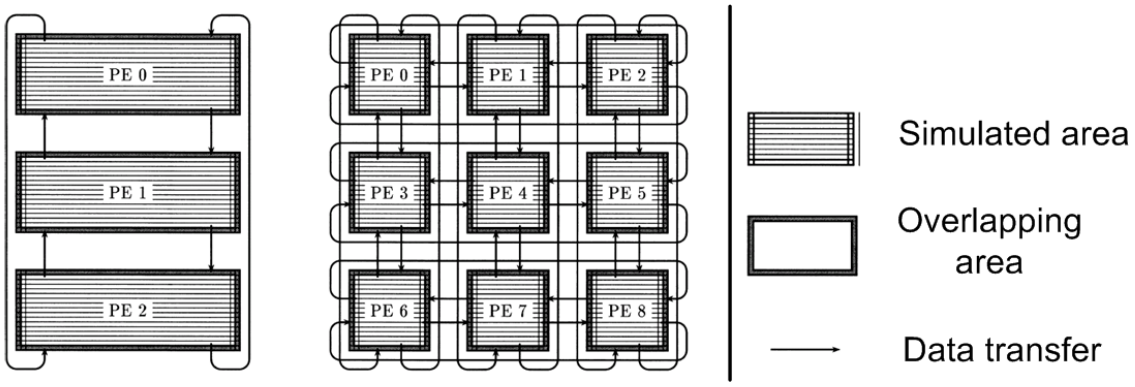

Figure 4.25: Illustration of domain decomposition in one dimension (left) or two dimension, where the lattice is decomposed into sublattices that are distributed to processors (PE) for simulation (from Worsch (1999), modified).

However, some general aspects can be stated as discussed in Worsch (1999): With domain decomposition, the lattice is subdivided into several sublattices, where each sublattice contains a contiguous set of cells. In each timestep, the serial execution algorithm then is applied to the sublattice on different processors (several lattices might be processed on one processor). For each sublattice, the local configuration should be directly accessible, which may require that each sublattice is extended by sublattice boundary cells for the the provision of corresponding state according to the size of the neighborhood. If required, the global configuration has to be composed from sublattices after each time step, e.g. if needed for output. In general, domain decomposition gives rise to performance gains, if the overhead induced by the synchronization of memory access and/or transfer of data between processors is outperformed by the gain through parallel execution of the transition function. Concrete characteristics depend on specific characteristics of hardware and models, but generally a simulation implementation aims at the avoidance of processing overheads and avoidance of latency of processors (Worsch, 1999). A general consideration is that the amount of interprocess communication rises with the granularity of decomposition and the size of state and neighborhood. Moreover, the simpler the computations of the local transition function (e.g lookup tables) the higher expected latencies due to communication. Locality of interaction allows the reduction of latencies, e.g. it is possible in each timestep to calculate the transitions for inner cells first, for which local configurations are complete, while local configurations for boundary cells are transmitted, which are executed last. Further, sublattices are overlapping such that the local transition function is calculated redundantly by neighboring sublattices, such that data exchange is not necessary at every timestep, but more calculations are involved (see (Worsch, 1999) for details).

At a general level, the application of methods of parallel discrete-event simulation (PDES) appears particularly considerable, when inhomogeneities, e.g. spatially or temporally varying computational workload within the lattice or heterogeneity of processors are expected. In contrast to classic CA, PDES is relatively involved with issues of heterogeneous computational workloads since it deals with structurally heterogeneous models and asynchronous time flow. Since PDES is more general than CA, PDES can generally be

used for the execution cellular automata. Investigations however are rather isolated efforts focusing on particular models, methods and hardware (e.g. Liu (2010), Muzy et al. (2008), Jafer and Wainer (2007)). E.g. Muzy et al. (2008) proposes the application of the idea of activity tracking to the simulation of CA, where at each timestep computations and are limited to those cells that actually may change state, other cells are ignored. Based on the assumption that at each timestep only a small subset of cells needs to be considered and that the set of cells in the subset is relatively constant (memory allocation etc.), this may lead to efficiency gains with respect to computation compared to exhaustive computation. Activity tracking however is based on the possibility to identify relevant cells at the base of their transition function at each timestep. Also, the application of general PDES simulation mechanisms (i.e. time-warp) has been considered, where different processors are not synchronized at each time step, but at the time of real interaction between cells of the corresponding sublattices (Liu, 2010; Jafer and Wainer, 2007). However, depending on the concrete approach, there may be considerable synchronization overhead, since processors need synchronization information at each time step in order to avoid causality errors (conservative methods) or there is the need to store part of the trajectory of each sublattice in order to roll back the state in case of causality errors when synchronization is more relaxed (optimistic methods). In general, the feasibility of PDES depends on the assumption that sublattices can be identified with relatively few interaction, which depends on the characteristics of specific models and the distribution of computations on hardware and has been illustrated at specific models and implementation.

As the presentation of prototypical models above has illustrated, there are common restrictions and extensions to the classic CA modeling paradigm that allow for optimization or that come along with loss of performance. The representation of the transition function in form of rules reduces calculations to a search in a rule table (e.g. counting rules can efficiently being encoded into exhaustive rule tables). This allows for efficient computation of the transition rule, if the rule table fits into memory. Further, the smaller the number of possible states, the more information might be stored in a machine word. In particular boolean CA allow for the optimization of the number of bits needed, which can be exploited for efficient calculations, but also for reducing the amount of data to be communicated between processor and memory or between processors. A further type of CA is that of partitioned CA[20], where the representation of the state of a cell is partitioned such that each partition provides exactly that portion of a state that is needed by one neighbor for calculating the transition. This can be used for highly efficient implementation of LGA (Worsch, 1999).

Thus in general, CA fit the needs for digital simulation, in particular when time and space are discrete. Restricting CA gives possibilities for optimization, in particular when the number of possible states and the size of the neighborhood is limited and few computations are required. Further, if behavioral properties are known in advance, these can be used for optimization.

Tools for Simulating CA

Any GPL or powerful simulation-oriented DSL can be used to implement CA models and depending on features of the language, hardware and compiler, efficient implementations may be implemented. Against the background of a variety of different motivations, breadth

[20]Please note that these CA are not equivalent with "partitioning CA".

of the concept of CA and technical considerations of simulation efficiency, it appears as the logical consequence that a variety of tools has been developed with different characteristics (see Worsch (1999)). The overview Worsch (1999) distinguishes two basic kinds of tools with respect to user interface[21]: first, tools are either given as shared libraries that can be used with GPL, and second tools offer DSLs in order to be able to apply optimizations with respect to execution time of simulations. Typically, tools offer textual concrete syntax for model implementation as "writing programs in the usual way (Worsch, 1999)."

Whereas GPL-based tools use the corresponding tool support of GPL, DSL-based models are typically compiled into source code of a GPL, which is the compiled and linked for execution. There is a variety of tools with different characteristics ranging from tools that are designed and optimized for the simulation of rather specific types of models (e.g. GOL, boolean LGA) to tools that impose few restrictions on models, but which then have limited support for optimization. Some tools are designed to support automated exhaustive exploration of rule sets of simple CA (DDLAB, LCAU), others even provide special hardware (CAM-6, CAM-8, see Toffoli and Margolus (1986)) for efficient simulation of simple CA models. Typical restrictions are a fixed number of dimensions (typically 2) and Euclidean grid, however the neighborhood is typically unrestricted, often given as an invariant vector of offsets. Different tools offer different types of predefined boundary conditions (e.g. fixed, cyclic). The restriction to a limited number of possible states (from 1 bit, to often 256) is typical, but often different registers (variables) can be used to give some structure to the modeling of state and to store the value of state variables of a different type. Some tools offer the possibilities for structuring such as C "struct". Typical DSLs offer the means of imperative programming languages for the definition of local transition functions or the local transition function is given as sets of rules, that resemble rule tables. For probabilistic modeling typically explicit calls to a random number generator are available, but some offer higher-level constructs, such as explicit annotation of alternatives with corresponding probabilities. Some packages offer the specification of block-rules (e.g. Margolus neighborhood) and registers/variables allow the specification of partitioned CA (the local transition function only reads one specific register/variable of each neighbor). Some tools offer specific variations of the classic CA paradigm, e.g. time as global variable (temporal heterogeneity), or the usage of the coordinate of the cell (spatial heterogeneity) and the explicit specification of asynchronous update.

In conclusion, there is a variety of tools which reflects the heterogeneity of pragmatic aspects, conceptualizations and technological considerations. Although the simplicity of the classic CA paradigm is considered a fundamental property, which is necessary to implement efficient simulators, the design of tools and the corresponding extensions and generalizations reflects that it is common to sacrifice computational efficiency for the sake of degrees of freedom and explicitness with respect to the conceptualization of models. However, it appears that the provision of DSLs is closely related to the optimization of simulation efficiency where possible, such that for optimization, modelers need not deal with the intricacies of parallel programming and a compact low-level representations state. As models are typically investigated in isolation, the integration of CA models with other models or coupling different CA models for compositional modeling is typically not being supported.

[21] The considered tool/languages are: CAMEL/CARPET, CASIM, CDM/SLANG, CELLULAR/CELLANG, HICAL, SCAMPER/CAL, CAM SIMULATOR, CAT/CARP, CELLAS/FUNDEF, CEPROL, LCAU, SCARLET/SDL, CAPOW, CDL, CELLSIM, DDLAB, PECANS/CANL, SICELA (Worsch, 1999)

4.4 CA for Macro-scale Modeling of Environmental Processes

4.4.1 General Properties of Macro-scale CA

According to the interdisciplinary character of EMS, micro-scale CA models in physics generally fall into EMS, e.g. fluid flow. However, in contrast to micro-scale modeling, CA in EMS are widely considered as a paradigm for modeling processes at a scale that corresponds to the macro-level or above in physics. However, there is the general notion that observed properties of systems at a great scale emerge as a result of interaction of relatively simple elements at a smaller scale and there are considerable perceived qualitative similarities with respect to observed fractal patterns, self-organization, critical behavior etc. For unambiguation, the two levels of CA-based modeling at the macro scale are further referred to as "local" - corresponding to small scale - and "global" - corresponding to great scale (aggregate level).

Indeed, the general issues of the use of differential equations and the issues related to numerical integration, complicatedness of formulation and absence of analytical solutions apply to macro-scale modeling as in the case of micro-scale modeling (see Chapter 4.3). Further, the general properties of small-scale CA modeling great scale phenomena namely the ease of integrating fluctuations and the straightforward explicit combination of different processes, generally motivate the use of CA for macro-scale modeling. At a general level, as micro-scale modeling, CA modeling at the macro-scale is widely perceived as a conceptual framework for original scientific reasoning, rather than a mere computational tool. However, there are considerable differences that come along with different requirements for supporting tools.

Generally, there is abroad range of phenomena and corresponding CA models that fall into the category of macro-scale models, which - besides scale - have in common that a cell of the CA is reasonably perceived to represent a portion of space that has some internal structure and which interacts with spatial neighbors (Hogeweg, 1988; Gregorio et al., 1999). Thus, macro-scale CA models might be models, where the structure and behavior of cells is directly prescribed by the macro-scale phenomenology of physics, e.g. a variable of the cell state represents its water content in terms of mass (not particles) and the transition is derived from macro-scale equations (not vice versa, see Gregorio et al. (1999)). However, the level of abstraction and theoretical background might be considerably different, as proposed in Tobler (1970), where the state variable of a cell represents the human population that resides in a cell. Tobler (1970) proposed CA as the natural framework for investigations that naturally aligns with the famous "[...] first law of geography: everything is related to everything else, but near things are more related than distant things (Tobler, 1970)" with the goal to achieve simplicity. Thus, CA are used to investigate systems that are not (exclusively) dominated by the laws of physics.

Most considerations about the specifics of macro-scale CA modeling indeed generally also apply to micro-scale modeling of physical processes, but apparently the degree to which properties affect methodological and technical aspects appears to justify separate consideration. Typical applications of macro-scale CA encompass phenomena such as vegetation dynamics, surface and subsurface hydrology with transportation processes, soil erosion processes, wild fire behavior, population dynamics, disease epidemics, urban growth, landslide dynamics and lava flow (Itami, 1994; Gregorio and Serra, 1999). The corresponding spatial and temporal scale of models might be quite different (e.g. spatial resolution of $\approx 10^{-4}$ meters to $\approx 10^{4}$ meters).

However some common properties of macro-scale CA models are typical:

- The cells have an explicit reference to geo-space.
- There is a richness of observations at the phenomenological level (e.g. geodata allowing the characterization of empirically based local structural and behavioral properties).
- The system is perceived as being composed of spatially correlated interacting processes.
- There is uncertainty with respect to the interplay of processes, although single processes might be relatively well understood.
- Systems are perceived as being open systems, thus they are typically not in global equilibrium, and there is explicit input at all spatial levels between local level and global level.
- Models are typically meant to reproduce quantitative aspects of systems measured at the global level, but also at the local level, such that it can be used for management purposes (e.g. decision support)

The general formal properties of macro-scale CA and how these correspond to classic CA is subject of some publications in different disciplines (Itami, 1994; Bandini et al., 2001; Couclelis, 1985; Dewdney, 2008; Hogeweg, 1988). Although the application of CA for macro-scale modeling is apparently motivated or at least inspired by the findings that are related to research of computational properties and micro-scale physical modeling, the framework of classic CA, and the degree to which generalizations are typically applied in micro-scale modeling, has early been perceived to be to rigid as to allow for adequate modeling. Besides the relaxations that are widely present at micro-scale modeling of physical processes (see 4.3: most notably probabilistic modeling, asynchronous update), (Couclelis, 1985) argues, that there are reasonable arguments to relax basically any characteristic of classic CA at the macro level: the assumption of static and uniform neighborhoods conflicts with the perception that influence depends on the spatial configuration, which might variable in time and space with respect to influential characteristics. Further, the heterogeneity and variability of structural and behavioral properties of spatial units under consideration conflicts with homogeneity of rules and cell geometry. Further, it has recently been noted (Dewdney, 2008), that the cells in macro-scale CA models rather resemble "miniature programs" than finite-automata, due to structural and behavior richness.

Thus, generally an identification of common properties is not possible on formal grounds, which necessarily would lead to the identification of CA as more general paradigm (e.g. networks of automata). However, as with CA-modeling at the micro scale, it is evident that with CA-based modeling at the macro-level in EMS, formal considerations must be set in context of pragmatic aspects for characterization. In the following, some common characteristic aspects CA at the macro-scale are presented at examples, that illustrate common motivations for extending the notion of CA, and corresponding formal properties, which however are limited by some pragmatic considerations, thus provide restrictions to widespread universal notions of CA[22].

[22]This indeed contrasts with many research efforts in the field that aim at the identification and implementation of novel extensions and generalizations of the notion of CA.

Specific common properties of macro-scale CA

This section illustrates some common properties of macro-scale CA models at some level of generality. Concrete examples are given in the following Chapter.

Layers and spatial hierarchies A common property of macro-scale CA is the need to the conceptualize the state and transitions according to the perceived processes involved. A common pattern for structuring the state description is to divide the set of possible states into *substates* . Each substate is associated with a particular process or a number of processes, if several processes depend on the substate or if it is used for interaction. Interaction shows as the usage of the same substates within the different processes, where at least one substate that is an input in one process must be the output of another. In conformance to more conventional descriptions of Dynamical Systems, a substate is typically modeled by means of a variable. Accordingly, the conceptualization of the local transition function might be structured into parts, where each part typically works on a subset of variables, either modeling the processes in isolation or the interaction between them.

A common approach to conceptualizing different processes is a "multiple -lattice - approach", similar to that in micro-scale modeling, where each process corresponds to a lattice (see Figure 4.26, see (Bandini and Mauri, 1999)). The variables are typically thought of being organized in layers, such that each lattice is thought of being associated with a number of layers and each layer might be associated with several processes[23]. Corresponding to the different properties of processes, each process might be defined at the base of a specific neighborhood and/or timestep. The interaction between processes is modeled at the base of local transition, thus spatial dependencies between processes are possible at the base of neighborhood relationships.

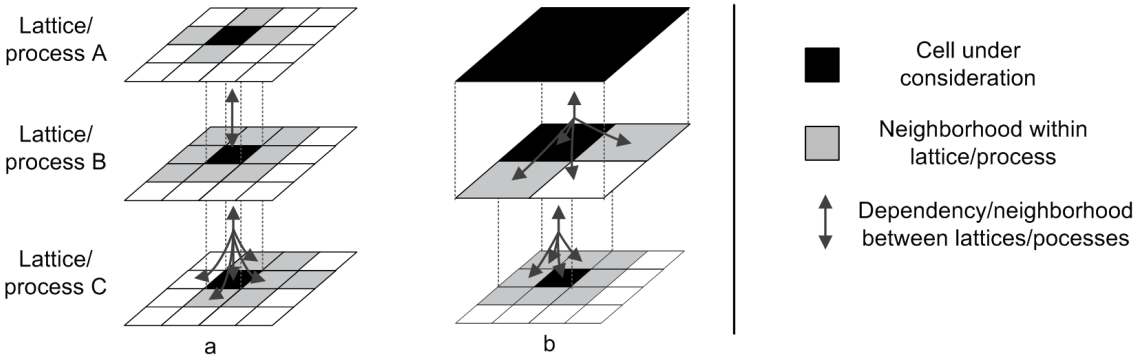

Figure 4.26: A spatially homogeneous layering with geometrically identical lattices (a) and and lattices with spatial hierarchy (b).

As in GST in general, the conceptual ordering of systems along a compositional hierarchy is generally a common approach to deal with complexity in macro-scale modeling (Wu and David, 2002). A frequently considered variant of multiple lattices that particularly aims at the explication of differences of spatial scale are *hierarchical CA* (see Bandini and Mauri

[23]Please note that in CA literature, a lattice may be referred to as "layer". To avoid confusion with the layer concept of GIS the term "lattice" is used analogously to micro-scale CA modeling. Although the motivation for multiple lattices is slightly different here (see Chapter 4.3).

(1999); Sonnenschein and Vogel (2002)), where the CA is organized in different lattices with different spatial, and possibly temporal granularity. Each cell of a coarse grained lattice is perceived to be related to those cells at a finer-grained level that occupy the same space. Whereas the relationship of spatially homogeneous layering is "local-local", the relationship between levels in hierarchical CA is that of type "local-global": the finer grain-cells provide input for coarse grain cells, which may involve aggregation, and the coarse grain-cells provide the same input to all cells of the corresponding lower level. Indeed at each level, usual CA evolution may take place.

Related processes might be perceived as causally ordered, in that one process is "more fundamental" than the other, insofar that one precedes the other, e.g. transport (e.g. fluid flow, information dispersal) precedes action (e.g. chemical reaction, decision), from this follows a serial execution of parts of the transition function. Further, processes might be perceived as evolving in parallel, which corresponds to parallel execution of all parts of the transition function, or they might be intermingled, leading to a fine-grained interaction patterns, where processes interact at a higher frequency as the timestep, e.g. when different spatial parts of the CA evolve one after another (e.g. critical behavior). Here different parts of the transition function are applied iteratively for subsets of cells or iteratively calculate intermediate states until the final state is reached.

Spatial Zoning At the macro scale, different processes are perceived to have different zones of influence and properties of the zones of influence are considered to be dependent on the local properties which might be variable with time. Since zones of influence are typically considered at the base of neighborhood, neighborhoods are perceived to be different for different processes and local configurations, thus there are typically several different, possibly time variant, neighborhoods for each cell.

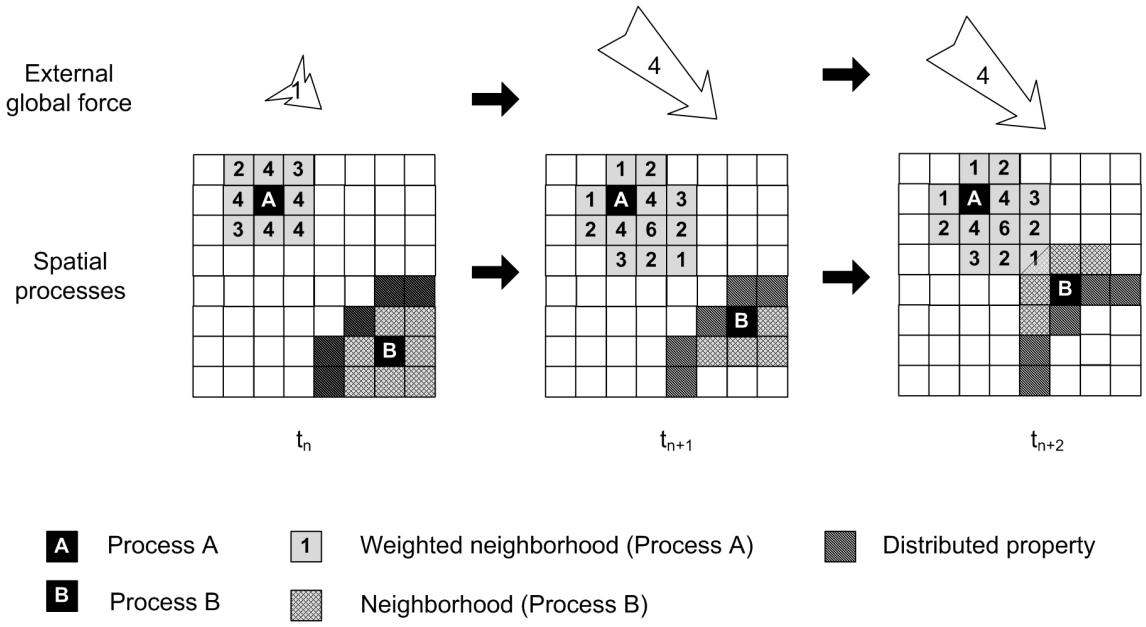

Figure 4.27: Illustration of spatial zones as neighborhoods: different processes (A,B) have different neighborhoods that might depend on different spatially distributed or global properties and change according to theses properties.

Figure 4.27 illustrates a configuration, where different processes proceed according to specific rules and dependencies. A typical property of macro-scale CA is that processes are influenced by both distributed properties that are perceived through local configurations and global forces, which are forces that act on all cells (not necessarily processes) in the same way (e.g. a homogeneous force field). Global forces might be perceived as external forces or they are a function of the global configuration of the CA. The characteristics of the spatial influence might vary according to the external force (see process A), where the shape of the neighborhood varies with the directional external force. Neighborhoods might have an internal structure that reflects spatial dependence of influence (e.g. distance), but also dependence on spatially distributed properties (e.g. barriers) and compound properties (e.g. gradients, angle between cells). As zones in general, the internal structure of neighborhoods may be represented by means of weights that reflect the quantity of influence.

Inhomogeneity and Temporal Variability of Spatial Entities The geometry of spatial units is typically not regular to the degree the lattice of a CA is and spatial configurations may change with time (see micro-scale modeling). In macro-scale CA modeling, it appears to be self-evident to explicitly consider the irregularity of observed geometries, if these are correlated with behavioral properties; e.g. the mass of a rock influences collision behavior (e.g. rocks in a landslide Avolio et al. (2009)). There are two general approaches to consider heterogeneous geometry: first, the geometry of the lattice might be irregular (see *Asymmetric CA*, Sonnenschein and Vogel (2002) or Geographic Automata Systems, Benenson and Torrens (2006b)) and second, following the local-global modeling approach irregular shapes are approximately formed from compounds homogeneous cells of the lattice.

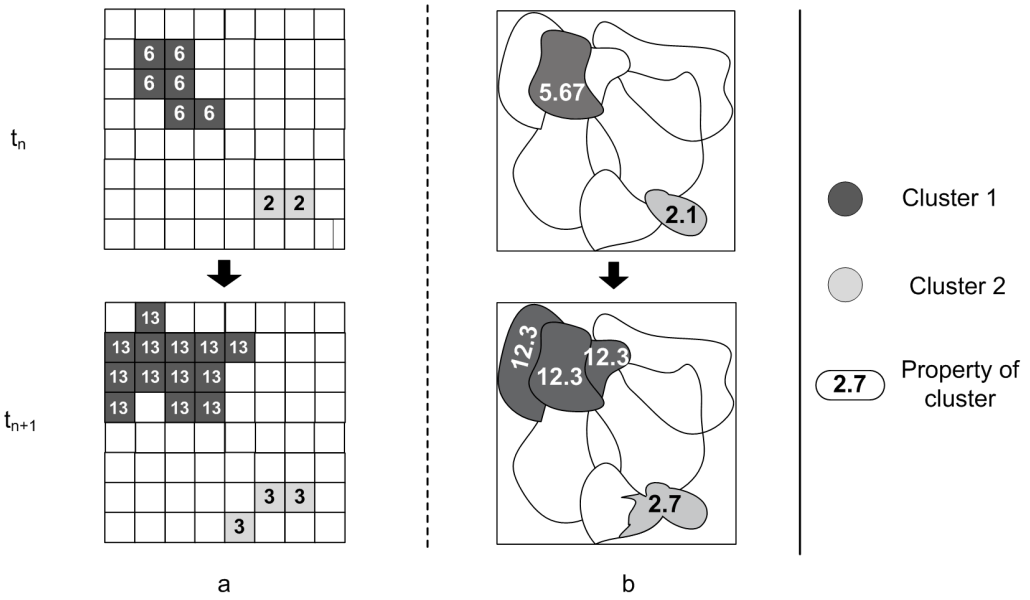

Figure 4.28: Illustration of geometrically heterogeneous configurations within a geometrically homogeneous lattice (a) and a geometrically inhomogeneous lattice (b), with numbers indicating a property that depends on the heterogeneous entity (e.g. area).

Figure 4.28 (a) illustrates the case where spatially heterogeneous entities are formed by means of clusters (gray) of regular cells. Clusters may have a global property, which depends on properties of its constituent cells (i.e. area of the cluster). These cluster properties might influence the behavior of cells insofar the members of the cluster might exhibit behavior different from others or that the neighboring cells take into account properties of the cluster. Whereas the formation of the cluster might be an emergent property defined along the lines of well-researched mechanisms, e.g. through local segregation or aggregation processes, the cluster properties have to be explicitly considered, thus have to be formalized and calculated during simulation as an aggregation over the set of cells within a cluster, if clusters are time variant.

Figure 4.28 (b) illustrates a corresponding case with explicit irregular lattice geometries. Here, the additional degrees of freedom leads to gain in precision as cluster properties can be calculated more accurately and the transition function is formalized directly on the heterogeneous entity, not on a compound construct. However, if the geometry of clusters is variant, the change of geometry has to be considered. First, variance can be treated similarly as in the homogeneous case, such that clusters are formed from geometrically invariant basic shapes and similar mechanisms of treatment of clusters apply. Second, basic geometries are variable: generally GIS provide sets of operations that handle operations on sets of geometries (e.g. Clementini set operators merging, adding, subtracting etc. geometries). However the application of such dynamism to M&S, in particular the definition of new geometries, has to the knowledge of the author not been considered generally yet.

Spread, Multi-step Transitions and Asynchronicity Spread processes within classic CA framework and in the area of micro-scale CA modeling conceptualize spread processes such that the target (cell) receives the spreading entity (e.g. particle) from the source (cell), such that spreading is formally dependent on the local configuration of the target cell. However at the macro-scale, spread processes are often considered from the perspective of the source, such that spread is considered as being dependent on the properties of the local configuration of the source cell. Please note, that source-dependent spreading can be considered in form of target-dependent spreading in two steps, like LGA (see Chapter 4.3.3), where the target cell is chosen within the local transition function of the source cell and the transition function of the target cell then simply "accepts" the spreading entity. This however requires that source cell is in the neighborhood of the target cell, and that the target cell can be denoted appropriately. Great neighborhoods, and time and space variance make this approach complicated to realize and requires explicit consideration of space.

Figure 4.29 exemplifies a spread algorithm ("minimization of differences"), where quantity (e.g. fluid) is distributed to cells with less quantity, such that differences are minimized (from Gregorio et al. (1999)). It consists of the repeated calculation of the local average quantity and the removal of cells with above average quantity, until all neighboring cells are below average, to which finally the quantity is distributed. Figure 4.30 illustrates a typical model of a spread process with some inherent randomness. Starting from a source cell, a process starts a random walk that might be conditioned on some spatially distributed property (e.g. roughness, slope), such that at every step of the walk the new target is selected probabilistically. The random walk ends, when a condition is satisfied, e.g. maximum length reached, target found etc.

A slight variation of such random walk enables the modeling of spatial discontinuities

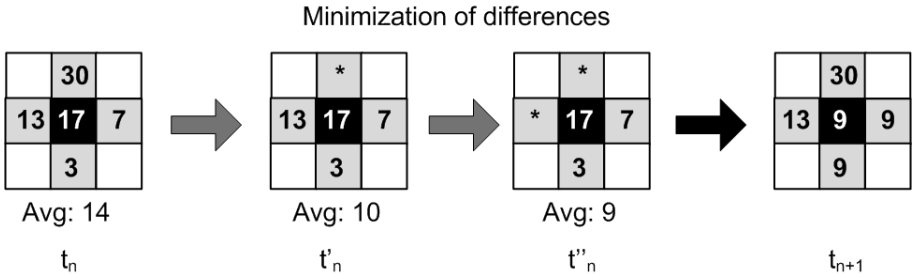

Figure 4.29: A spreading algorithm that distributes matter such that differences between cells are minimized: the average quantity is calculated and cells with above average quantity are removed iteratively until all neighbors are below average to which the quantity is distributed (from Gregorio et al. (1999)).

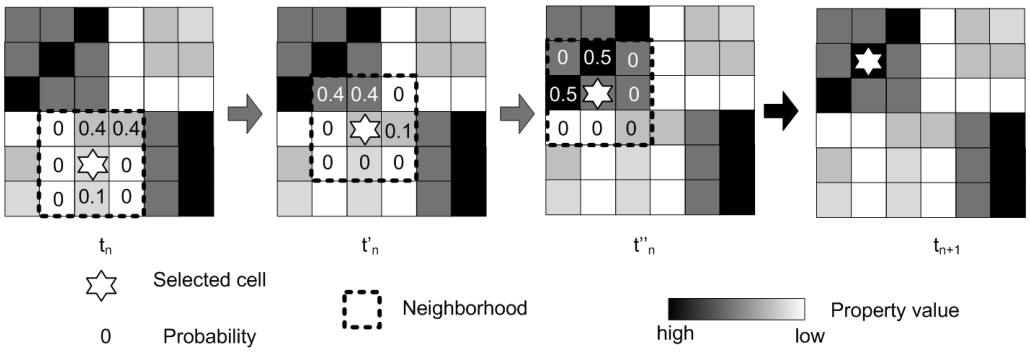

Figure 4.30: An exemplary random walk, where a source cell is selected from which a random walk takes place that is conditioned by some distributed property and ends at some stopping condition (e.g number of steps, target found).

that are considered to be well aligned with the idea of macro-scale CA modeling: if the neighborhood is enlarged, the movement might take a distance of more than one cell per step. By this, the process may overcome barriers that would not be traversable by spatially continuous movement[24]. Such discontinuities are frequently considered, e.g. in models of fire spread (see example in Appendix D), where different modes of movement (e.g. combustion and flying sparcs) are considered at a rather abstract phenomenological level. Figure 4.31 illustrates this, where *process 1* passes a barrier.

Although explicit spreading aligns with intuitive conceptualization of the dependencies on the local configuration of the source cell, considerations of target cells however are more difficult to handle, e.g. maximum quantities in a target cell, since different source cells might target the same cell. Figure 4.31 illustrates possible strategies to deal with this issue, a) all influences of all source cells are considered and affect the state of the target cell as in isolation, (b) conflict is resolved in that a source cell chooses a new target and (c) only one source cell is considered.

The above examples further illustrate the property of common macro-scale CA that the transition function proceeds in distinct interrelated steps, with intermediate states, and often proceeds iteratively. Computations might be complicated, such that any method

[24]This is behavior which is particularly difficult to include in models based on differential equations.

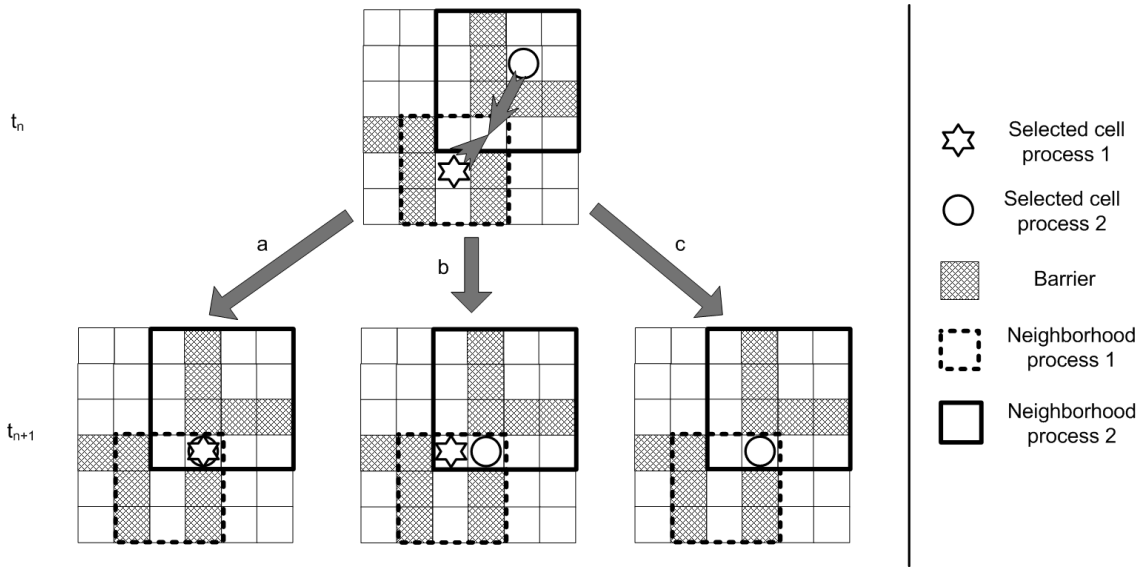

Figure 4.31: A spread process with enlarged neigborhood and conflicting transition.

to structure the description of computations from software development can be applied, structured programming and high-level formalisms, e.g. Petri nets. The use of structured transition functions align well with separate consideration of distinct interrelated processes with a serial ordering, where one process precedes the other or develops at much higher speed, e.g. a physical interpretation could be that a process relaxes to local equilibrium before the other process acts (e.g. two phase flow Gregorio et al. (1999), SOC). Serial execution of parts of the global transition function also align well with asynchronous behavior, where different parts of the CA evolve at different speeds, possibly under the influence of different processes, where only a subset of cells "is active" insofar that their local transition may lead to state changes (e.g. fire spread in large areas). Different speeds of processes might be considered as explicit choice of times of update following an discrete event approach. Further, the general issue of synchrony of events is subject of discussion in macro-scale modeling (Hogeweg, 1988): on the one hand it seems to be the reason for emergence of interesting behaviors, on the other hand it conflicts with the perception that processes are not equally present and synchronized in the whole space.

4.4.2 Method and Pragmatics

As mentioned above, there is reason to propose and reject any extension or generalization of the classic CA paradigm for macro-scale modeling. It appears that reasons for extensions and generalizations basically can be found in the perceived heterogeneity of observations at the macro-scale, for which it seems usually possible to find good reasons for direct explicit consideration. Reasons for rejection are mainly the loss of simplicity and considerations related to simulation efficiency.

Dewdney (2008) states although the great majority of scientist is more familiar with the use of differential equations, the usage of CA is widespread and closely related to CA as a means to generate "insight", thus they are perceived as a means to discover new structures and processes, not to bear resemblance. This point appears to be pivotal, since it appears evident that with the power of ever more generalized notions of CA, it is possible to directly

represent any spatially extended system with the means of CA. However, to speak of an insight is only possible when observations emerge that are not explicitly encoded into the rules. Again, as in the case of micro-scale CA modeling, the defining feature of CA based modeling appears to be the notion of simplicity in combination with some kind of emergence, such that not explicitly encoded observations can be reproduced, in particular spatial behavior and patterns (Hogeweg, 1988). However, at the macro-scale CA appear to be provide a framework with which it is relatively straightforward to encode assumptions about the system under study and "alternatives to these assumptions present themselves as straightforward extensions (Hogeweg, 1988)." It appears however, that except for purely physically-based processes the theoretical underpinning is relatively uncertain, such that validation basically is based on empirical validity and scientific discussion. Although there is considerable application of macro-scale CA modeling in these areas, the validation of CA in terms of theoretical underpinning is a major challenge.

Methodologically the usage of insight generating CA is typically such that - given observed behaviors and spatial patterns, macro-scale laws or empirical laws - CA are constructed based on intuition, where much inspiration is drawn from mechanisms discovered at CA models at the micro scale (Hogeweg, 1988; Bandini et al., 2001; Dewdney, 2008). Epistemic uncertainty typically relates to the absence of theoretical underpinning or the knowledge about the relevance of a wealth of perceived causal structures. Thus, a key ingredient of studies is the exploration of alternative structures under different conditions (Dewdney, 2008). This means that knowledge and intuition are the base for assumptions that are tested by varying an assumed mechanism in form of CA models and evaluate expected and actual behavior, which involves traversals of parameter space for a given mechanism (see Dewdney (2008); Hogeweg (1988); Bandini et al. (2001); Itami (1994))[25]. However, Torrens (2009) states that "issues surrounding calibration and validation of cellular automata models are chief among challenges facing future research in this area."

More practically, this means that models are typically constructed such that a variety of alternatives can be explored automatically by means of automated parameter studies. Much of practical work is related to the analysis of geodata, which is basic form of data input and output. Both, the efficient implementation of simulators for CA and computer-based analysis of data pose considerable technical challenges which are the major drivers for the development of corresponding tool support. The increasing availability of geodata at the macro-scale and corresponding GIS technology motivates the integration of geodata in M&S studies that refer to technical issues of integration (see Chapter 3.4). As there are many available observations and there are different interacting processes under consideration, the number of possible states is relatively high and the representation of state typically structured, such that the structure of state reflects the structure with respect to the modeled processes; in particular, when compared to models at the micro-scale.

Thus, when comparing to the micro-scale modeling framework

[25]The exploration of alternative structures as modifications of a mechanism distinguishes what (Hogeweg, 1988) refers to as "data oriented models", which formally are CA models, but the transition function gives transition probabilities for the possible configurations that are derived by statistical analysis of observations.

4.4.3 Tools and languages

General Aspects of Simulating Macro-scale CA

Since Cellular Automata have a wide range of applications and provide means to demonstrate the parallel processing facilities of any general-purpose M&S tool, there are - besides those tools considered specific for micro-scale CA modeling - numerous tools that offer support for CA-based modeling as extensions to those tools. However, efficient automatic parallelization is the better possible the more restrictions are imposed on the models. As presented above (Chapter 4.3.5) locality, restrictions on the number of possible states, size and variability of neighborhoods, simplicity of local transition functions and the explicit consideration of "activity" of cells gives possibilities for optimization, given a parallel hardware with a number of processing elements. Whereas tools for micro-scale modeling typically exploit restrictions of sizes of states and the invariance of neighborhood, macro-scale tools rather exploit mechanisms of event-driven simulation, which are generally tailored towards relatively variable structures, explicit time-flow and the notion of activity, thus the omission of unnecessary calculations (which is a particular phenomenon of difference equations and classic CA). However, a particular aspect that has to be considered in any approach is the presence of global variables and parameters that are typically used to model external influence, and communication among distant cells. These and the typically relatively complicated calculations of transition functions severely limit possibilities of optimization by means of parallelization and are subject to research, which is not further considered in this thesis, although being relevant in general.

Tool support for CA modeling in EMS

General aspects that apply to already presented tool support for M&S (Chapter 3.3), EMS (Chapter 3.4) and micro-scale CA modeling (Chapter 4.3.5) generally apply to tools for macro-scale CA modeling:

- CA tools for EMS should and often do incorporate GIS functionality, as an extension of GIS or an extension of a M&S tool.
- CA tools come as extensions of GPL that may allow for multi-paradigm modeling, but incorporate general issues related to GPL as modeling languages.
- CA tools provide a DSL, but the tools are typically monolithic (see below).
- CA should and often do provide support for automated experiment series.

Particularly present in literature are those tools that provide CA modeling as an extension of one of the DEVS formalisms (see Chapter 3.3.3) that typically perceive a CA model as a special kind of DEVS-model that is composed of homogeneous DEVS sub-models (e.g. Posse et al. (2006), Shiginah (2006), Wainer (2006)/CellDEVS/CD++, Filippi and Bisgambiglia (2004)/JDEVS, Praehofer et al. (1993)/STIMS-CLOS). These works basically serve the purpose of showing the integration of CA models and others at the base of DEVS, the power of parallel DEVS simulators, and how relatively high-level descriptions of CA can be transformed into DEVS. Particularly related to this work are the works related to the language metamodeling tool $ATOM^3$, since these generally apply a meta-model based approach to provide a formalism for the specification of simple CA models and use

a corresponding method (graph transformations) to generate a DEVS model. The studies however focus solely on the use of graph-grammars and do not contribute to CA modeling above that.

As technical issues have been considered in the more general case already, the next section presents some that aspects add some background for the description of the modeling language considered in this thesis, that particularly focuses on the integration of GIS and CA modeling tools. Although there is a variety of tools, only some of them consider the specifics of EMS in that they aim at the provision of some expressivity related to the conceptualization of transitions, but they basically provide the means of imperative languages and corresponding control structures (see Chapter 3.2.1). Thus, although there is expressivity with respect to structural aspects (e.g. neighborhoods, boundary conditions etc.) often combined with some special purpose expressivity (e.g. for activity tracking or modeling the evolution of time), issues of GPL apply to the formulation of transition functions.

```
Program     ⇒ Declarations Cells Agents Observers
Cells       ⇒ Grid Attributes Init Behaviour
Grid        ⇒ grid Axis [ , Axis ]* ;
Axis        ⇒ Index .. Index : Id <.> Id : Boundary
Index       ⇒ IntConstant | Id
Boundary    ⇒ static | cyclic
Attributes  ⇒ cells { [Type Id ; ]+ }
Behaviour   ⇒ behaviour { [Type Id [ = Expr ] ; ]* [Instruction ]* }
Instruction ⇒ Assignment | Cond | ForEach | Switch
Assignment  ⇒ Id = Expr ;
Cond        ⇒ if ( Expr ) Block else Block
ForEach     ⇒ foreach ( Type Id in Set ) Block
Switch      ⇒ switch ( Id ) { [Case ]+ [Default ]
Case        ⇒ case CaseVal [ , CaseVal ]* : Block
Block       ⇒ Instruction | { [Instruction ]* }
```

Figure 4.32: Partial grammar of the CAOS language for modeling CA models.

Figure 4.32 illustrates the language elements of a typical DSL for modeling CA at the base of its grammar (*CAOS*, Grelck et al. (2007)). The partial grammar (see (Grelck et al., 2007) for detailed illsutration) shows that the language embodies high-level constructs for the specification of lattice (*Grid*), boundary condition (*Boundary*), and allows the specification of the local transition function (*Behavior*) at the base of statements similar at the level of imperative programming: typed variables (*Attributes*), assignments (*Assignment*) and control structures (e.g. *Cond*, *ForEach*). Only some DSLs have been developed with the aim of some generality with respect to specifics of macro-scale CA modeling in EMS that do not recourse to low-level concepts of GPL for modeling transition functions, but which derive corresponding language elements from domain analysis. Notable examples are *PCRaster* (Karssenberg, 2002) and *SELES* (Fall and Fall, 2001) (see below).

GIS-based tools for CA modeling GIS typically impose a layered perception of systems, where a layer corresponds to a specific spatially distributed observation. There is a straightforward correspondence between CA and GIS in that GIS provide specific types of layers with regular and irregular geometries and each geometry is associated with a number of typed, typically numeric, attributes that represent the characteristic of a geometry.

Rasters[26] are a specific type of layers that divide the represented space into a uniform grid of rectangles, typically squares. In contrast to other types of layers, usually one raster represents one observation (e.g. temperature), thus several observations are organized as a collection of rasters.

The correspondence of rasters and CA is straightforward in that one raster represents one variable that represents the state of a cell, thus CA with several state variables correspond to an equal number of rasters for the same spatial extend and same resolution. Although it might be possible to store several attribute values in one cell typically there is one raster dataset for one variable and one point in time.

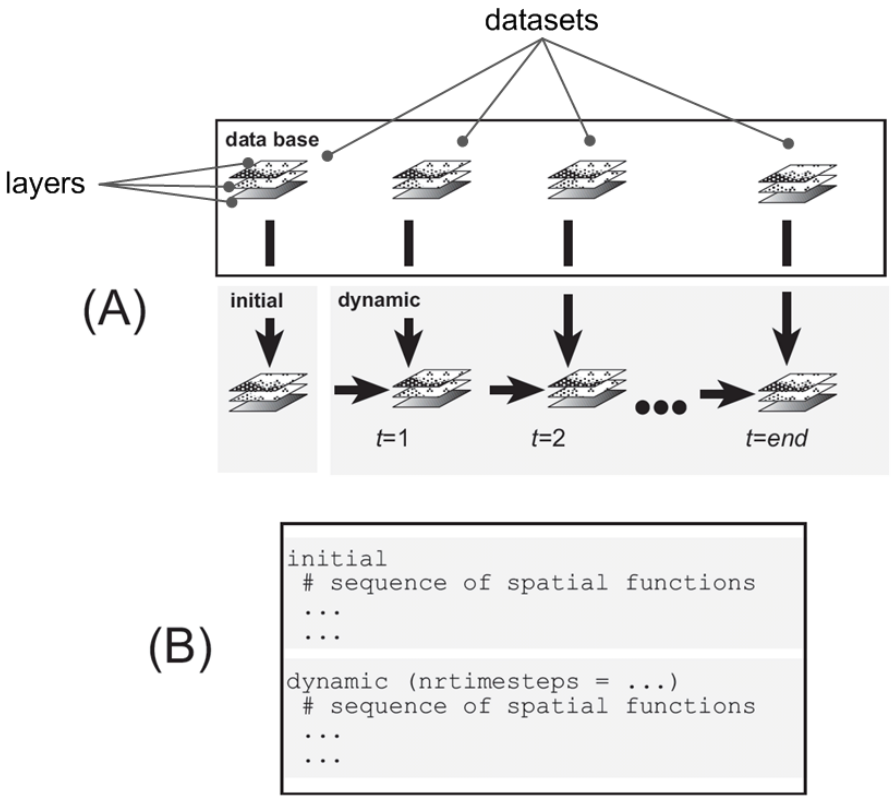

Figure 4.33: Illustration of GIS-based dynamic modeling where the system is organized in layers (A) and dynamics are conceptualized as the iterative application of spatial functions to the corresponding layers represented by GIS datasets (from Karssenberg (2002), modified).

Figure 4.33 (A) illustrates this perception. Dynamic spatial models are perceived as performing operations on sets of spatial datasets and, by this, produce new spatial datasets that correspond to the layers of the model. These operations - alongside operations for retrieval and storage of geodata in the database - are typically accessible via the API of GIS tools and is often used in a programming like fashion using GPL (B). However, some GIS tools (*ArcGIS ModelBuilder*, *Sextante Modeler*, *MapWindow Modeler*, see Marchionni and Ames (2009)) provide higher-level DSLs for the specification of corresponding models

[26]The term "raster" widely corresponds to the term "GridCoverage" in the context of OGC/ISO standards and denotes a GridCoverage with a square cell geometry (see Chapter 5.3).

in form of workflows (see Figure 4.34). A workflow denotes the application of predefined operations (i.e. Filter) to geodatasets. An operation produces a geodataset. Operations reside in a corresponding repository and datasets reside in the database of the GIS tool that provides a GUI for modeling.

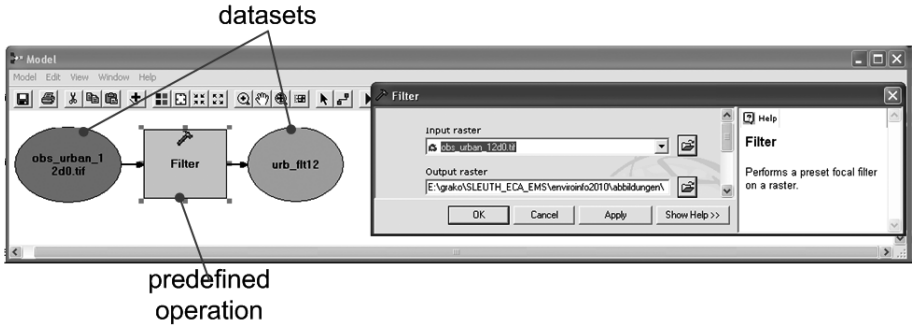

Figure 4.34: GIS-based modeling workflow with ModelBuilder.

PCRaster (Karssenberg, 2002) is a tool for EMS that provides a language (*prcalc*) with particular focus on macro-scale CA for modeling hydrodynamical processes in rocks and on the surface. It is a combination of GIS and simulation software. It includes more than 100 functions that operate on raster datasets of the form *ResultMap* = **function** (*InputMap*$_1$, ..., *InputMap*$_n$), where "InputMap" refers to a raster. These functions are the language elements of a script-like DSL that can be used to describe the flow of operations the make up the state changes of the CA. Besides 2-dimensional uniform square lattices, *prcalc* provides a 3-dimensional spatial type that explicitly allows the representation of formations of rocks and soils. Four types of functions are distinguished: functions that only operate on the state of a single cell, functions that operate on a contiguous neighborhood, functions that operate on all cells and functions on a arbitrary predefined neighborhood. All functions determine the next state of a single cell under consideration and are executed at each fixed timestep serially. There is support for automated experimentation.

SELES (Spatially Explicit Landscape Event Simulator) is a tool and DSL that has explicitly been built in order to facilitate the simulation-based exploration of landscape dynamics. Particular focus is on incorporating language constructs that allow the representation of concepts of landscape change (Fall and Fall, 2001). A model is conceptualized at the base of raster layers, global variables (structure) and events (dynamics). Figure 4.35 (a) illustrates the hierarchical ordering of events and related entities: an event instance is associated with clusters that have been formed during the course of an event instance and each cluster is composed of cells, called *active cells*. Events encapsulate the description of behavior of event instances, that follow a common pattern: (i) a set of cells is chosen probabilistically, (ii) it is probabilistically decided if an event is establishes in the chosen set of cells, (iii) cells proceed with transitions, (iv) it is probabilistically decided when a process spreads and where it spreads to.

Figure 4.35 (b) illustrates that for the different steps of events and spreading processes different information is available. The availability of information is organized in four contexts: the *global* context holds a-spatial global information, the *spatial* context additionally contains information of the spatial layers, the *active cell* context contains information about an active cell (local configuration) and containing cluster, the *recipient cell* additionally contains the local configuration of target cell. Thus, underlying CA

is asynchronous and SELES is specific in that it explicitly supports clusters and spread transitions.

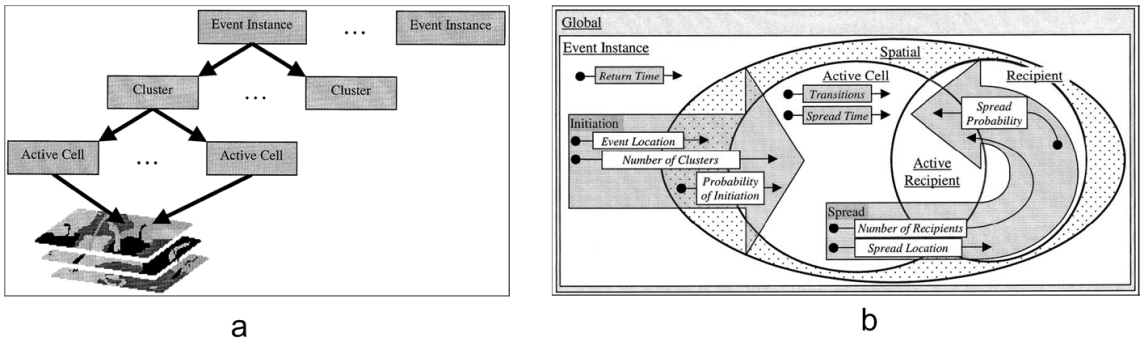

Figure 4.35: Illustration of the hierarchical ordering of events and the control and information flow within SELES models (from Fall and Fall (2001), modified).

Although both tools (SELES and PCRaster) provide some generality with respect to EMS, they are both tailored towards a specific application domain and they are monolithic insofar one has to use the runtime system of the respective tool for simulation. Within tools, both provide facilities to reuse and couple existing model descriptions, however none provides support for multi-paradigm modeling. However, besides the direct use of geodata, both tools incorporate a variety of special (spatial) functions, such as for calculating distances, areas, angles, random numbers and (spatial) statistics, that are specific for geospatial analysis. Further, both tools offer some kind of assignment and control flow specification (e.g. loops). Thus in general, although the mentioned DSLS offer a relatively high level of abstraction, compared to CA modeling tools with similar universality with respect to possible CA models, they are certainly not "small" in the sense of minimality of language elements and not overly regular in the sense of orthogonal language constructs. E.g. spatial functions require different types of input than arithmetic functions. Both tools offer runtime support in form of graphical user interfaces and dynamic visualization.

4.5 Conclusions

In conclusion, it has been shown that the paradigm of CA is particularly related to an exploratory approach to investigate assumingly complex systems. CA are perceived to provide relatively simple and intuitive mechanisms that are generally able to reproduce any behavior. However, in practice there is no common formal notion of CA, in particular in the field of modeling at the macro scale. Nonetheless, the notion of CA is used frequently. It appears that the notion of CA refers to some notion of homogeneity, but more to a notion simplicity and explorative method of knowledge discovery, where CA-based mechanisms provide small-scale mechanisms for observed great-scale observations. Thus, it aligns well with the considerations of model-based science, where great-scale observations, if somehow regular and precise (e.g. empirical laws), may be perceived as meta-properties for which models or abstractions of it provide causal mechanisms. Micro-scale CA based modeling of physical processes provide a wealth of prototypical mechanisms for general observations (e.g. pattern formation) that are perceived as sources of intuition, or - in terms of model-based reasoning - as a source for analogical reasoning.

Whereas the source mechanisms of micro-scale physical modeling are well-documented and of relative simplicity, the transfer to the macro-scale in practice appears to take place in quite different conceptual framework, although it is also referred to as CA. The heterogeneity of notions and corresponding tools indeed is related to computational aspects and the application of computational and methodological characteristics of CA-based modeling to a variety of investigations. The heterogeneity of notions of CA is a fundamental issue, in particular as CA are typically used to investigate complex systems, where small differences generally may have arbitrary consequences - parameter values, initial states, implementations and variations of mechanism. The widespread use of extension mechanisms of GPL to support CA-based modeling indeed supports aspects of automation and multi-paradigm modeling and the provision of access to efficient simulation technology. However, it does object transparency, since at least the central part of the description - the local transition function - is arbitrarily complicated and does not reflect mechanisms under consideration, but data and control flow of programs. This also applies to DSLs that provide GPL-like functionality for specification of the local transition function[27]. The few relatively "domain-oriented" external DSLs are rather specific and bound to a specific tools, such that reuse of corresponding models is limited to this tool. There is no DSL that aims at providing generality across domains of EMS, that is not monolithic, that provides support for geodata and which does not use constructs of GPL for transition modeling.

Although the issue of parallel execution CA simulation is intrinsic, the common features of CA based modeling severely limit traditional ways of parallelization, basically through relatively complicated local transition functions and non-locality of communication (global variables, clusters). Recent developments rather aim at the avoidance of unnecessary computations by means of parallel discrete-event simulation. This however is an active field of research and not subject of this thesis. In the following, the language-centered approach to EMS is presented, before the Environmental Cellular Automata Language is presented, which is a DSL that aims at providing the above mentioned characteristics, based on a specification that follows the language-centered approach.

[27] The widespread provision of GPL-like structures might be explained by the fact that most DSLs compile to a GPL, which gives the possibility to provide a powerful language and keep the transformation simple.

5 The Language-centered Approach for Tool Support for EMS

5.1 General Considerations

The language-centered approach (LCA) denotes an approach to implement modeling tools for EMS. The most fundamental characteristic is that the different aspects of M&S studies as identified in Chapter 3.6 are explicitly represented by means of corresponding models.

Figure 5.1 gives an overview of aspects and corresponding DSLs defined and used within this thesis.

- *simDescription*: a rudimentary prototypical DSL for the description of experiments and experiment series developed for the purpose of demonstrating and investigating the coupling of DSLs for experiment and system modeling. SimDescription is a heavily simplified and adapted version of *ExpL*, which is a DSL for experiment description developed by Frank Kühnlenz (see Kühnlenz et al. (2009)).

- *ECAL*: a DSL for the description of CA models for EMS developed and described within this thesis (see Chapter 6).

- *mobileAgent*: a rudimentary prototypical DSL for the purpose of investigating the coupling of DSLs for system modeling developed in this thesis and described in Theisselmann et al. (2009a).

- *GISDSL*: a rudimentary prototypical language defined within this thesis as a sub-language of simDescription for the description of workflow-like GIS-based analysis of data generated by simulation models.

- *simCore*: A rudimentary DSL for the description of couplings between models developed withing this thesis. The language metamodel of simcore encompasses basic concepts that are used to define DSLs (e.g. ECAL) that are used to define coupled models.

- *timeSeries*: A rudimentary DSL developed in this thesis that allows the definition of models whose trajectory is given by time series data.

Please note, that the DSLs used in this thesis are to be understood as "proof-of-concept" and indeed do not provide comprehensive functionality as indicated by Figure 5.1, thus functionalities are limited to those basic aspects described below.

Generally, the approach investigated within this thesis is that of realizing a "superformalism", where an encompassing DSL is the composition of several coupled DSLs, following the approach that all concepts of coupled DSLs are part of the superformalism (see Chapter 3.5.1, Figure 3.5). Further, the approach followed is that of transformative definition of operational semantics. Thus, the meaning DSLs is given in terms of a transformation to

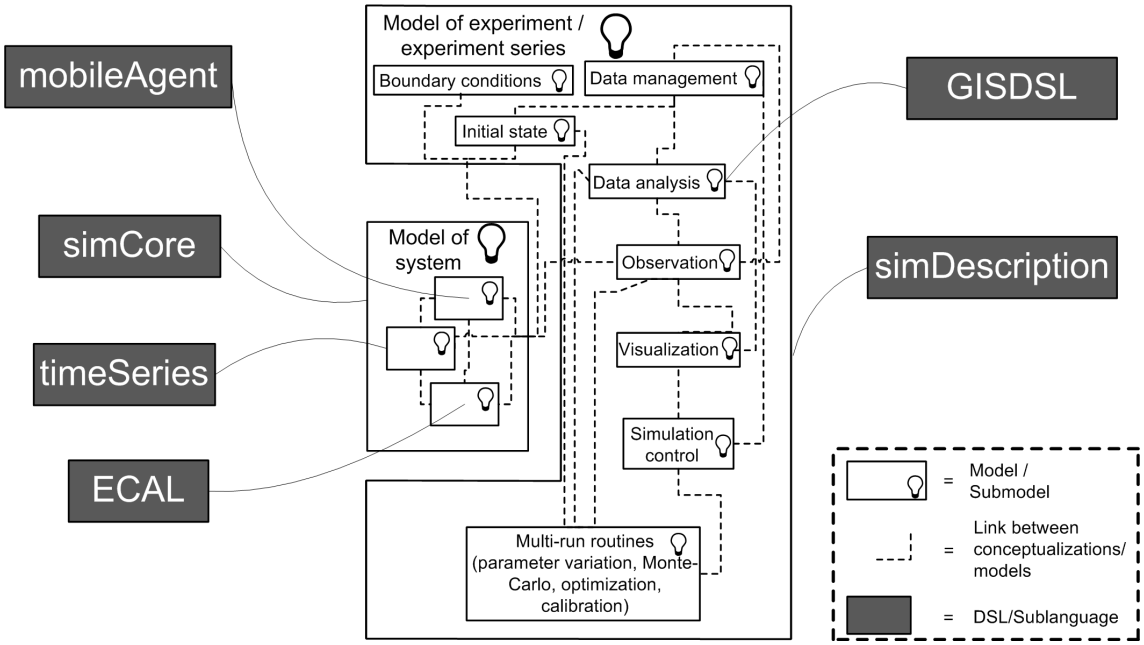

Figure 5.1: Illustration of LCA and how these correspond to the aspects: model, experiment and analysis.

another computer language that has a defined operational semantics. Given the existence of a variety of relatively universal simulation technologies based on GPL (see Chapter 3.3), this approach is generally tailored towards the reuse of DSL-based models with different low-level technologies with a common set of functionalities and different non-functional properties. This requires reasonable assumptions about common functionalities that form the semantic grounding of DSLs in the transformational approach.

GPL provide the most general semantic foundation for DSLs. However, with some restrictions a semantic base can be identified that reflects a considerable class of models of Dynamical Systems and there are various available implementations providing tested implementations of numerical approximation and synchronization mechanisms with different non-functional properties (e.g. optimizations). Based on the considerations in Chapters 3.3 and 3.4, the semantic base of the LCA considered in this thesis is combined discrete-event simulation, since it combines a general time-flow and synchronization mechanism for discrete-time models with state-of-the-art numerical techniques. Further, it is assumed that models are built following the idea of "model components", thus models are defined as entities, that might be models of systems, when used in isolation, or submodels, when used in combination with other models. This typically comes along the perception of a model being a "black-box" model that produces an output given an input, where internals are hidden. Although this imposes the severe restriction that there are no immediate feedback loops between such model components since transitions are atomic, it reflects the practice of EMS in particular, where reuse of models is envisaged (see Chapter 3.4.4, 3.4.5).

A further basic part of EMS is the consideration of geosaptial data processing. The use of ISO and OGC specifications in the field of GIS as semantic foundation is considered in Chapter 5.3 below. Since experiment specification and execution is not the focus of this thesis, no particular evaluation of these aspects is performed in this thesis.

5.2 The Realization of LCA with Metamodels and Transformations

Figure 5.2 illustrates the basic characteristics of the "proof-of-concept" implementation of this thesis. At the formalism-level modelers use a modeling tool to specify an technology independent specification of a model with no syntactical dependency to particular technologies at the framework level. For this, a modeling tool provides a set of DSLs (i.e. ECAL, GISDSL etc.) - coupled into one superformalism -, where the abstract syntax of DSLs is defined by an object-oriented language metamodel. The modeling tool that basically provides an editor and code generator for the DSLs, is itself based on language technology of MDE. This language technology encompasses a meta-metamodel and provides high-level facilities to specify modeling tools at the base of metamodels that conform to the meta-metamodel (i.e. TEF, OAW).

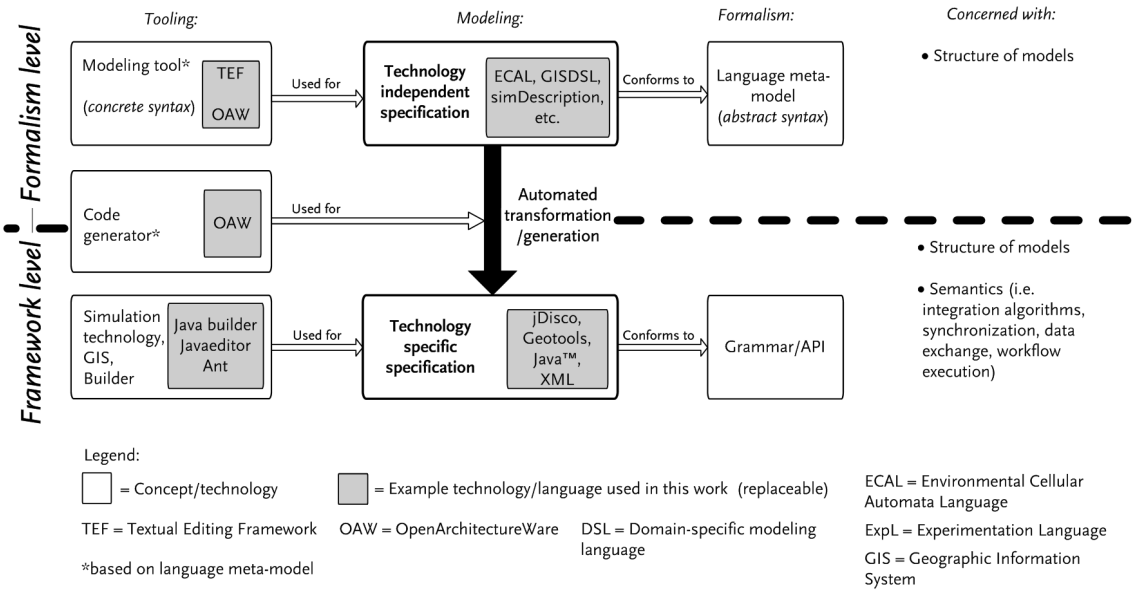

Figure 5.2: Metamodel-based implementation of LCA.

Besides the editor for modeling, the modeling tool provides the model transformations and code generation that produces code as input to tool support of the framework-level technology (i.e. compiler). Figure 5.3 gives an overview of basic technologies used at the framework level. Based on the Java programming language, the shared libraries *JDisco* and *Geotools* have been combined to form the shared library (*ECALibrary*), which provides the API against which the code generator generates code[1]. *JDisco* is a process-oriented discrete-event simulation library for combined modeling (Helsgaun, 2001), thus it provides discrete-event processes as the fundamental concept for modeling. *Geotools* is a library that provides GIS functionality as defined in OGC specifications (see Chapter 5.3 below). The prototypical implementation of the LCA within the tool ECA-EMS uses *Ant* as technology that executes experiment workflows encompassing the execution of experiment series and corresponding automated data analysis.

[1]Please note that ECALibrary is generally not necessary, but it has been introduced in order to provide the invariant code initially produced by the code generator for practical purposes.

All framework-level technologies take text files as input whose general structure is prescribed by a grammar. These text files - Java code (*ECALLibrary*) and XML (Ant) - are generated automatically by the modeling tool ECA-EMS, that also triggers the compilation of the Java code into an executable binary. The modeling tool is implemented with the Eclipse Modeling Framework (EMF) that provides the technologies *openArchitectureWare* (OAW) and the *Textual Editing Framework* (TEF) for the metamodel-based automated implementation of modeling tools. In particular EMF provides facilities to define and manage meta-models and derive modeling tools (editors, code generators) from them. For this OAW provides DSLs for the definition of metamodel-based code generation (xPand), model transformations (xTend) and their execution (OAW Workflow). *TEF* provides a DSL for defining an textual editor based on a metamodel, such that the editor is automatically retrieved (see Chapter 3.5.2).

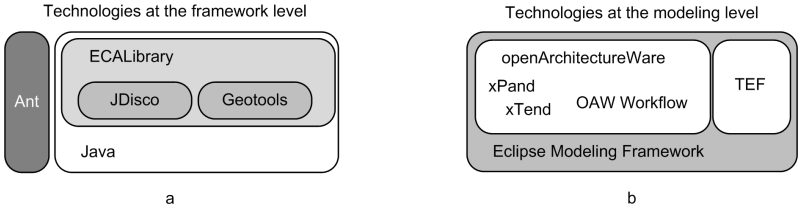

Figure 5.3: Technologies used at the framework-level (a) and at the modeling level (b) in the prototypical implementation of LCA constituting the modeling tool ECA-EMS.

5.3 ISO/OGC Specifications as a Semantic Base

The ISO TC 211 and OGC (see Chapter 2.1) define a series of interrelated norms and standards that aim at supporting the convergence of concepts onto which GIS tools are built upon, with the purpose of supporting the interoperability of GIS systems. Generally, interoperability must be based on the definition of common concepts, which provide the base for abstracting from technical detail of implementation according to the model-driven approach. Formally, these efforts adopt the method of language metamodeling as defined by the OMG that corresponds to the four-layer metamodeling approach (see Chapter 3.5.2). Thus, it appears self-evident to consider the metamodel-based integration of GIS and M&S. However, the specification of corresponding concepts are distributed over a considerable amount of documents (omg (2006), OGC (2003b), Miller and Mukerji (2003), Object Management Group (2007), ISO (2000), OGC (2001), OGC (2008), OGC (2006), OGC (2005), OGC (2004b), ISO (2003a), ISO (2003b), OGC (2003a), ISO (2002), ISO (2001)). Appendix A provides a graphical overview of basic concepts and relevant specifications.

Generally, ISO TC 211 adopts the object-oriented metamodeling approach by defining an object-oriented metamodel for the definition of spatial features - the *General Feature Model* (GFM). The basic concept of corresponding geospatial modeling is the *feature*, which formally is a pair of geometry and attributes that represents one or more real world objects. As a metamodel, GMF provides means for the object-oriented modeling of *feature types* of which features are instances, analogously to UML classes of which objects are instances, with the difference that features are meant to represent spatially extended

real world phenomena. The GMF uses UML class diagram notation for the specification which widely corresponds to UML meaning (some modeling guidelines are described in (ISO, 2002)). Figure 5.4 shows the kernel of the GFM.

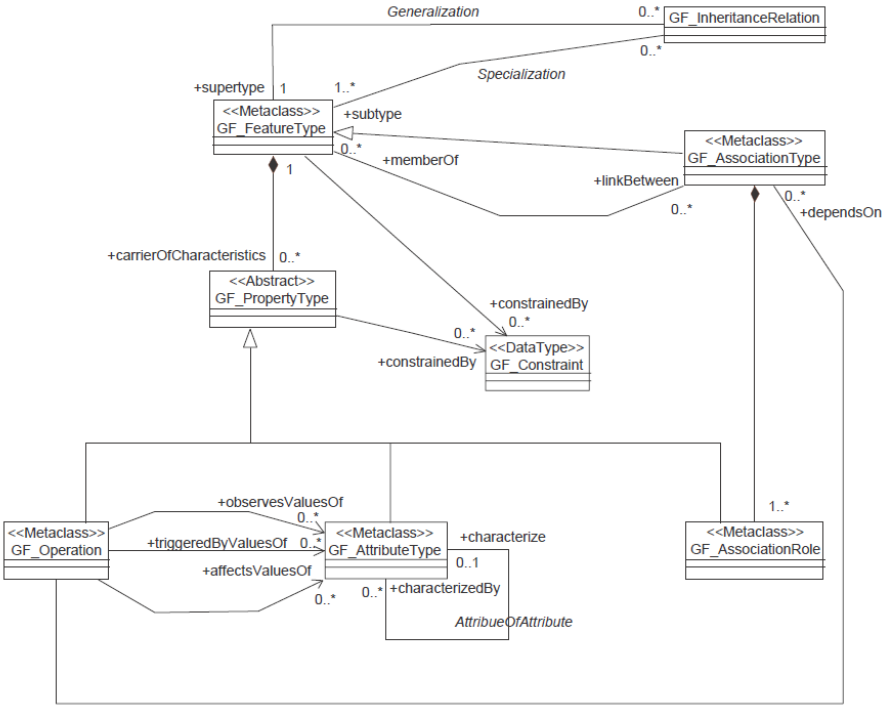

Figure 5.4: The kernel of the General Feature Model (from ISO (2002)).

In practice, the definitions of feature types are organized within UML class diagrams as representations of GFM models that define *application schemas* that correspond to specific applications (e.g. cadastre, river and road networks etc.) . An application schema defines the logical structure of geospatial data representing feature types and may define operations that can be performed on or with the data, thus it is a model of the data. However, application schemas are meant to address the logical organization of geodata, rather than the physical, thus they provide guidelines for the implementation of software that processes geodata. Since 2005 the OGC adopts the GFM as modeling language for modeling feature types instead of UML. OGC defines *Abstract Models* that define an "eventual software system in an implementation neutral manner. [...] The Abstract Model is a description of how software should work [and] represents a compromise between the paradigms of the intended target implementation environments OGC (2009)." *Abstract Models* are organized within documents that are refered to as *Abstract Specifications*. OGC *Implementation Specifications* describe how applications can implement abstract models at the base of specific technologies, such as Java and CORBA.

Conceptually, the integration of the GFM with metamodels of DSLs is generally possible, if some representation of the GFM technically integrates with the DSL-metamodel (e.g. GFM is translated into or mapped onto Ecore used in this thesis). However, except means for the definition of classification structures for features types analogously to class structures in UML, the GFM does not provide specific support for modeling geospatial features. In particular, it does not provide operational semantics. Semantics is given at

the modeling level by means of specifications (Application Schema, Abstract Specification and Implementation Specification).

Coordinate Reference System For example (ISO, 2003a) defines that spatial characteristics of a feature may be described by spatial attributes of type *GM_Object*. Geometries are modeled by means of subclasses *GM_Object* as a combination of a coordinate geometry and a coordinate reference system. Figure 5.5 illustrates the definition of spatial reference systems and that specifies (a) a *Coordinate Reference System* (CRS) is composed of a *Datum* and a *Coordinate System*.

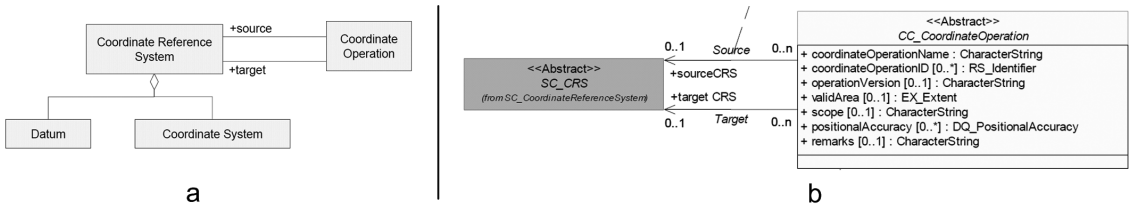

Figure 5.5: Abstract specification of spatial reference systems (from OGC (2004a), modified).

A coordinate system is a framework in which numeric coordinates can be used to determine a location. A datum specifies the relationship of a coordinate system to the earth with its curvature and irregular shape. Thus, with a CRS, locations of a coordinate system are mathematically brought correspondence to locations on earth. This allows the mathematical derivation of distances, angles and other geometric elements from coordinates and vice versa OGC (2004a). There exist a variety of well-defined CRS for different purposes and regions of the earth. A well accepted catalogue of CRS is that of the European Petroleum Survey Group (EPSG, http://www.epsg.org/) that assign identifiers (Spatial Reference System Identifier, SRID) to such systems, which are commonly used in GIS to identify coordinate reference systems. An important aspect of standardized CRS is that there are many predefined operations defined on them - e.g. transformations of one coordinate reference system to another - such that their specification is subject to standardization (see Figure 5.5, b).

GridCoverage A basic concept defined by abstract and implementation specifications ((OGC, 2006, 2001) is the *GridCoverage* as a subclass of *Coverage*. It is a feature type that particularly fits the needs of CA based modeling and which is used for data representation in this thesis. Figure 5.6 presents an abstract model (from OGC (2006)) that defines that a coverage - like other features - spans a spatial domain (*Domain*) and has a CRS associated with (*CRS*). But the essential property of coverage is to be able to generate a possibly different value for any point within its domain (operation *evaluate()*).

A grid coverage is a coverage that has a grid coordinate system which allows for addressing individual cells in the grid where individual cells are centered on the grid points (see Figure 5.6, b). In a grid coverage, only the cell values may change - the size and geometry never change. As one major form of representation of spatial fields, there are typical operations defined on grid coverages, such as the calculation of statistical measurements (see OGC (2001)).

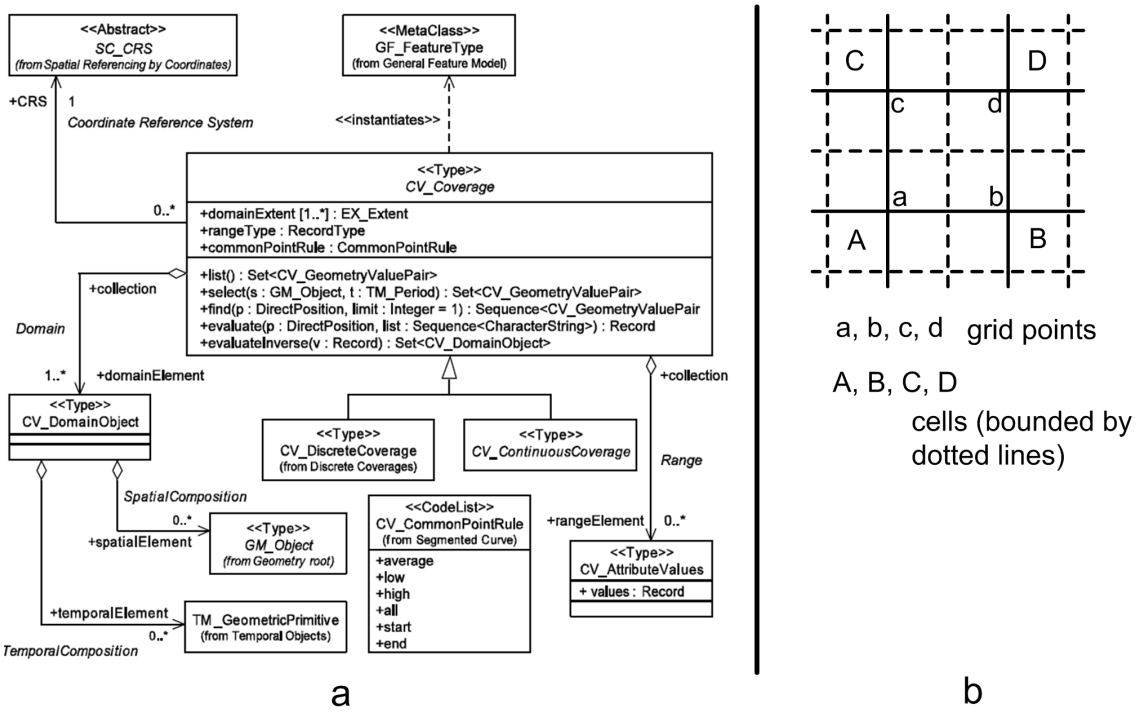

Figure 5.6: The abstract specification of coverage (from OGC (2006), a) and (b) illustration of a grid coverage (from OGC (2006), modified).

The above examples illustrate at examples how Abstract Specifications and Implementation Specifications provide concepts and general semantics related to processing geodata. Although the GFM is rather general with respect to structure and limited with respect to semantics, thus direct inclusion into LCA, the specifications at the modeling level provide abstractions of which one may expect to find corresponding existing implementations in the field of GIS. Although direct conformance to prescribed interfaces might not be expected - and different levels of conformance reflect this - it is assumed in this thesis that a common set basic concepts and functionalities may be found across typical GIS systems. These functionalities provide a semantic base for elements of DSLs.

5.4 Metamodels, Language and Model Coupling

Object-oriented metamodeling naturally provides the means to structure the description of abstract syntax by means of packages. Figure 5.7 presents the main packages that define the DSLs used in this thesis. The names of the packages correspond to the names of the DSLs. The package *SpatialDataHandling* encompasses some concepts related to GIS-based abstractions, *expression* contains a metamodel of typical universal mathematical expressions and computation statements (see below).

5.4.1 Experiment and Analysis

Figure 5.8 shows the basic classes of the metamodel of *simDescription* and *GISDSL*.

An experiment description, as an instance of metamodel element *Experiment* that is

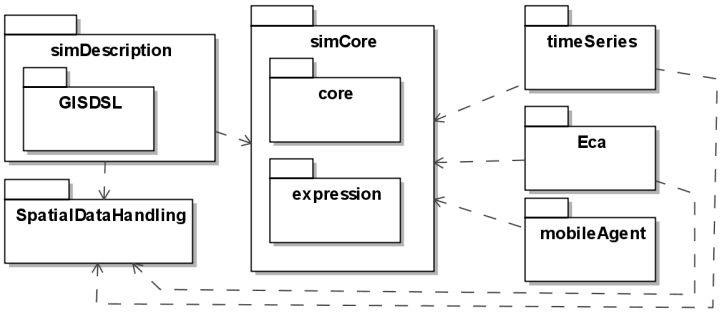

Figure 5.7: Overview of the package structure of the language metamodel of LCA.

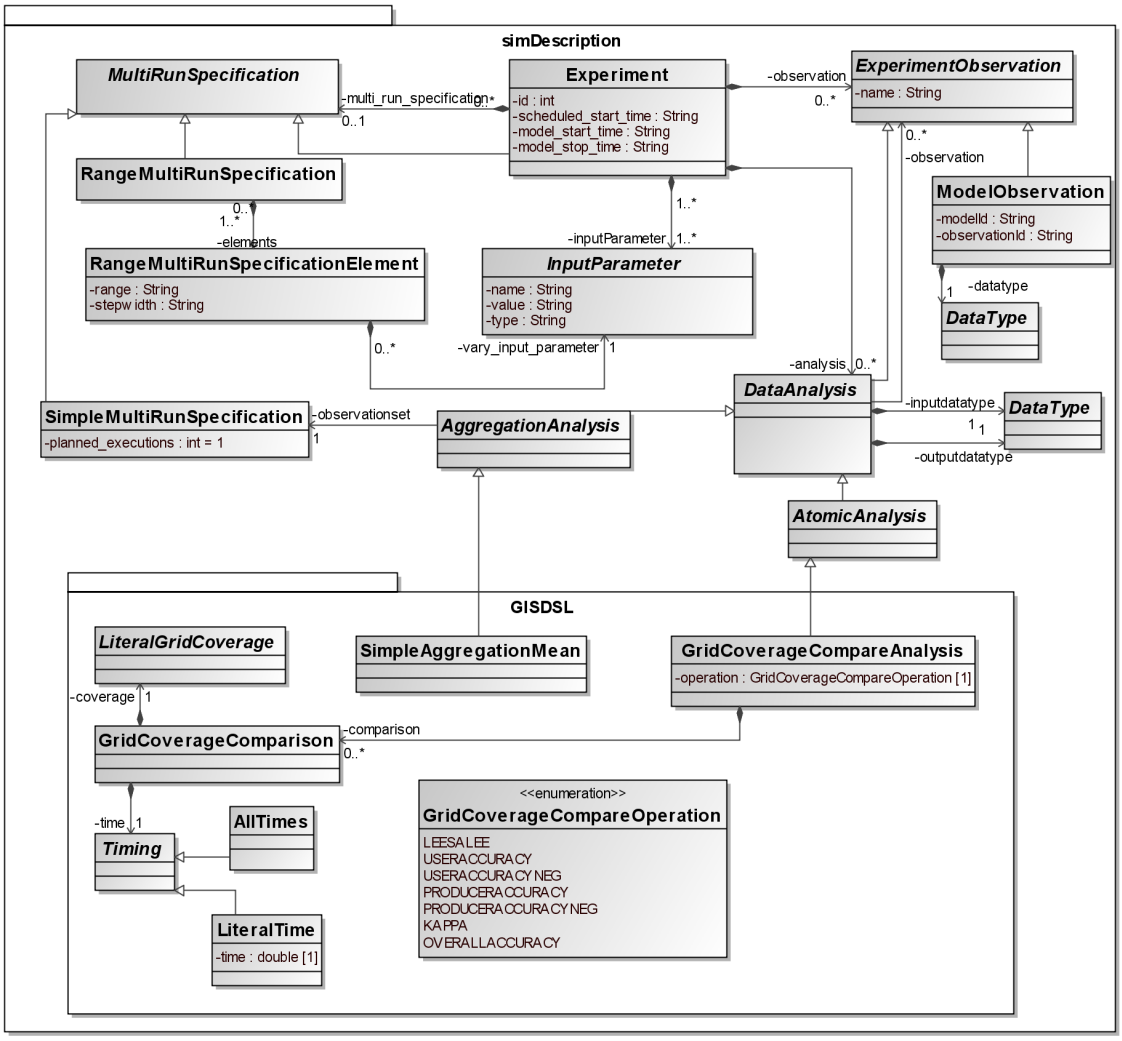

Figure 5.8: The kernel of the metamodel packages simDescription and GISDSL.

defined within *simdescription*, basically defines an experiment (series) by means of:

- Data that is made available to models (*InputData*).

- The specification of an experiment (series) that defines a (series of) initial states (*MultiRunSpecification*), e.g. by means of parameter variation (*RangeMultiRunSpecification*) or MonteCarlo simulation, where random number generator is initialized differently for each simulation run (*SimpleMultiRunSpecification*).

- the specification of observations to record and analyze (*ExperimentObservation*) that may be either connections to output ports of models (*ModelObservation*) or the results of a data analysis (*DataAnalysis*)

- the specification of a data analysis workflow (*DataAnalysis*).

Although very simple, the metamodel illustrates the straightforward coupling of abstract syntax of DSLs into one: with *DataAnalysis simDescription* contains an abstract element that prescribes that the DSL may contain concepts that allow other DSLs to define elements that perform an analysis on observations (association *observation*), where the analysis must define an input data type and an output datatype (*inputdatatype* and *outputdatatype*, see Appendix B, Listing 4 for an example). An experiment may encompass a number analysis descriptions and the output of an analysis might be input to another analysis, thus a chain of analysis steps might be defined. Further, two basic kinds of analysis are given: *AtomicAnalysis* which takes one input and produces one output and *AggregationAnalysis* that takes as input an series of experiments and aggregates over a set of corresponding observations. However, *simDescription* does only provide abstract classes, which means they cannot be instantiated by the modeling tool.

GISDSL is intended to represent DSLs that perform GIS-based analysis on geodata, such as common in GIS systems (see Chapter 4.34) and is tailored towards the application on the use case described in Chapter 6.2. Two kinds of analysis operators are introduced by the classes *SimpleAggregationMean*, which simply calculates the average value of the specified observation of a Monte-Carlo experiment, and *GridCoverageCompareAnalysis*, that performs an analysis on two grid coverages. Some predefined operators are available as used in the case study (e.g. *LEESALEE*, see Chapter 6.2). The specification of the *Timing* element denotes the times (times in the model) at which analysis is calculated (see Appendix B, Listing 5 for an example).

Indeed the coupling of DSLs requires that semantics fit. Since operational semantics is given as transformation, this cannot be ensured within the metamodel describing only abstract syntax. However, shared elements provide a way to at least partly align semantics at the level of abstract syntax. In the prototype, all DSLs share a common set of *DataType*, that contains some standard datatypes (e.g. for the reals, integers) with direct mapping to programming languages, but also the type *TypeGridCoverage*, that refers to ISO/OGC grid coverage, where the semantics is given by corresponding specifications. However, DSL-specific datatypes can be introduced, and when a transformation to common datatypes is provided, states modeled with these datatypes can be observed (see Chapter 6.1) .

Metamodel elements that are intended for reuse are organized within the *expression* package within the *simCore* package. *Expression* contains basic expressions that are commonly used in computer language-based model descriptions (see Figure 5.9). If language elements are reused directly in the definition of a new metamodel, e.g. by association or as a type of attributes, corresponding templates that define code generation might be reused too.

Listing 5.1 presents a code generation template (*expression*) that generates Java code ("Math.sin(Math.toRadians(...))") when applied to an instance of metamodel class *Sine*

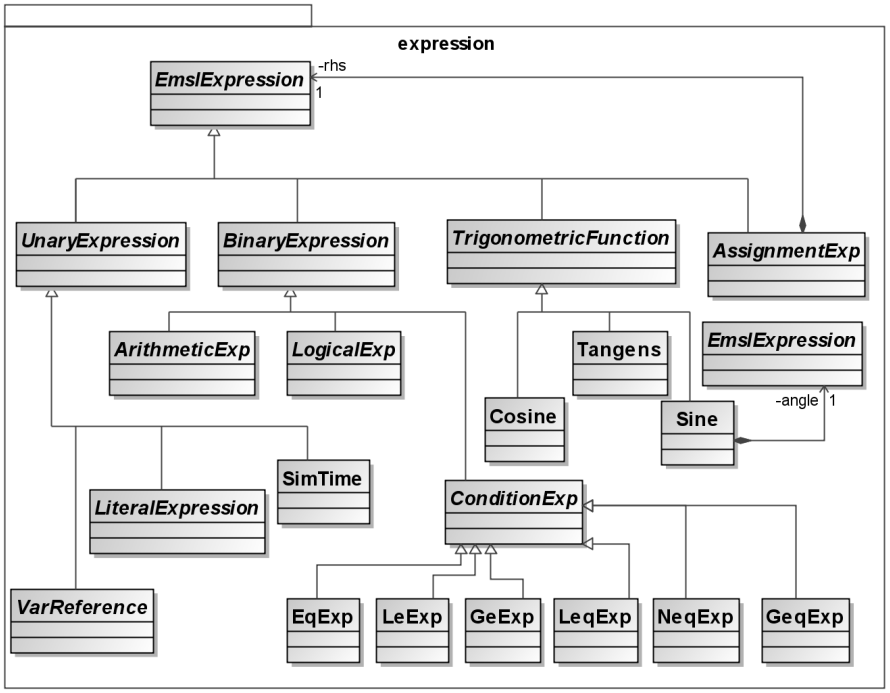

Figure 5.9: Some illustrating elements of the expression package meant for reuse.

from the *expressions* package (see Figure 5.9). This template can be applied to other DSLs that include *Sine* and target the same target technology[2].

Listing 5.1: A reusable code generation template in *xPand*.

```
1 «DEFINE expression FOR Sine -»
2   Math.sin(Math.toRadians(«EXPAND expression FOR this.angle»))
3 «ENDDEFINE»
```

5.4.2 Coupled Models

Figure 5.10 presents basic elements of the core package of the metamodel, that prescribes the basic structure of descriptions of coupled models.

The basic element is the *rootModel* that contains (specifications of) a number of *SubModel* instances (*model*) that are descriptions of models (*submodels*), and a number of specifications of the couplings between submodels (*coupling*). *rootModel* is a subclass of *Submodel*, thus it is formally possible to define a hierarchy of nested models. With this approach, the coupling and nesting structure of descriptions corresponds directly to the coupling and nesting structure of described models[3]. DSLs for modeling are introduced by subclassing the abstract metamodel class *SubModel*, which prescribes some properties.

[2]The template further illustrates the navigation mechanism of *xPand*, that iterates a model by recursively applying templates to instances of metamodel elements along the association and containment structure of a model specification (i.e. expand template *expression* for the model element that occurs when following the association *angle*). The complete code generation in the implementation presented in this thesis is realized with such templates.

[3]However, the nesting structure is not further defined and investigated further in this thesis, thus the following text relates to "flat" models with no nesting.

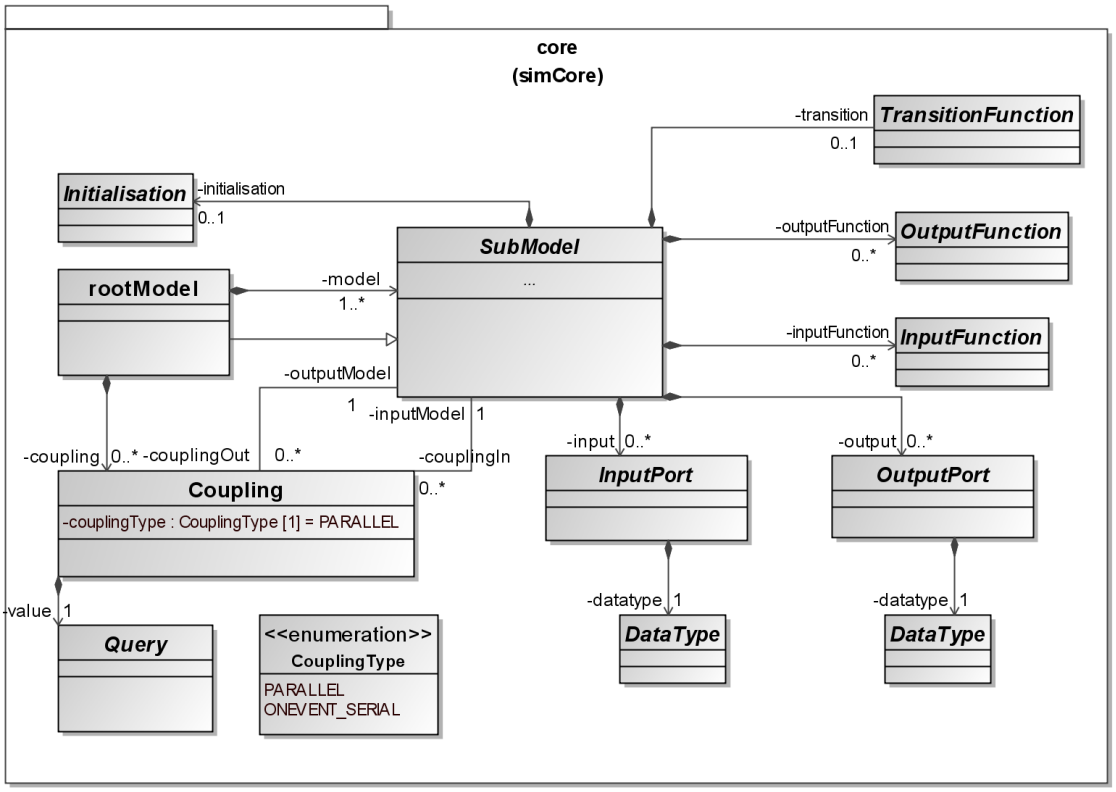

Figure 5.10: The kernel of the core package of the LCA metamodel.

According to the notion of "model component" (see Chapters 3.3.4 and 3.4.4) a DSL for the definition of models may define means to specify:

- *Initialisation*: the initialization of the model.
- *TransitionFunction*: the transition function.
- *InputPorts*: input ports used for coupling.
- *OutputPorts*: output ports used for coupling.
- *InputFunction*: mapping of values of input ports to the values of state variables.
- *OutputFunction*: mapping of values of state variables to values of output ports.

Figure 5.11 illustrates the system modeling DSLs defined in this thesis as subclasses of *SubModel*. The DSL *ECAL* modeled by *EcaModel* is further explained in Chapter 6.

The corresponding simulation procedure starts with the execution of *Initialisation* for all submodels, such that all models are in initial state. The subsequent simulation procedure basically follows the notion of discrete-event simulation: models proceed through time by scheduling their next event to a particular time in the future simulation time. However, models may be such that they do not explicitly schedule next times, but "subscribe" to another model, which means the times of events of the subscriber model are the event times of the provider model. There is no priority concept that prescribes the

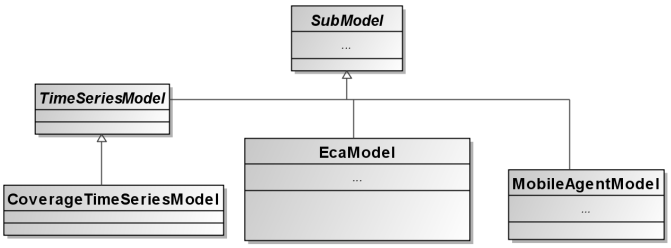

Figure 5.11: Definition of DSLs by subclassing *SubModel*.

order of model transitions for models that are scheduled for the same time, thus these are perceived to be executed in parallel. The execution of events follows a simple procedure: first, *InputFunction* is executed, then the transition function is executed and finally the *OutputFunction* is executed. The execution of coupled models is described in more detail below.

Each DSL for describing submodels has to provide means to describe the basic elements used for coupling: *InputPort* and *OutputPort*, where for both, DSLs must provide means to define a *name* as identifier and the specification of a *DataType*, thus ports have a name and type. The meaning of ports is that these hold input and output values of models at the time of simulation (see below) and output ports can be perceived as variables that store values that can change value by the application of *OutputFunction* of the containing model and which are read when *InputFunction* of models are executed. The couplings of models is specified at the means of a set of *Couplings*, where each coupling description describes the connection of one output port to one input port. Each coupling contains a reference to a *Submodel* (*inputModel*) whose input ports are connected to the output ports of a *Submodel* (*outputPorts*) as defined by a *Query* specification, that specifies the output port (examples are given below). One input port can only be connected to one output port, but one output port can be connected to many input ports, if permitted by the coupling scheme (see below).

The coupling scheme adopted in this work is oriented towards the simple coupling schemes used in practice of EMS that only considers the coupling at a discrete time base, thus time steps: the couplings basically follow the notion that a model "subscribes" to the output of another model which means that at the time of a scheduled event it reads the output port of a coupled model, such that corresponding values occur at its input ports. The coupling scheme considers two basic kinds of common approaches to component-based modeling in EMS and provides two corresponding language elements that indicate the type of coupling:

- Model components proceed in fixed time steps and read the input values of time $t-1$ to calculate the state of t. This type of coupling is indicated by *PARALLEL*
- Model components are chained, such that the transition of one model triggers the transition of the next model and each model is perceived to provide part of an overall transition function of the chain of models. This type of coupling is indicated by *ONEVENT_SERIAL*.

Listing 5.2 presents an coupling description as it appears in concrete syntax of *simCore*. Assuming that there are three models given with the identifiers *Model1*, *Model2*

and *Model3*, lines 2 and 3 define a coupling, where the output port *PortOut1* of *Model1* is coupled to input port *Port1* of *Model2* with parallel semantics (*Synchronization = PARALLEL*). Lines 5 and 6 define a coupling with on event semantics analogously between *Model2* and *Model3*.

Listing 5.2: Coupling description in *simCore*

```
1 ModelCouplings{
2     Name = Model1ToModel2, LinkModelOut Model1 InputModel Port1@Model2
          Synchronisation = PARALLEL
3     Value[ NumericQuery{SELECT [OutputPort(PortOut1)]} ];
4
5     Name = Model2ToModel3, LinkModelOut Model2 InputModel Port1@Model3
          Synchronisation = ONEVENT_SERIAL
6     Value[ NumericQuery{SELECT [OutputPort(PortOut1)]}];
7 }
```

In order to avoid inconsistent inputs, "ONEVENT_SERIAL" subscribers may only subscribe to one model and models with discrete time steps (*PARALLEL*) may not subscribe to *ONEVENT_SERIAL* coupled models and further, cyclic dependencies are not allowed. Algorithm 2 presents the corresponding basic simulation procedure for coupled models.

5.5 Conclusions

Based on the evaluation of existing technologies for EMS, M&S and technologies from the field of MDE, LCA has been proposed as a general framework in which DSLs for EMS can be developed by extending the metamodel. Although the prototype introduces rather severe restrictions on time flow and synchronization, a generalization within the limits of existing discrete-event simulation is possible. However, this might require explicit consideration of time flow and synchronization within the coupling language and the DSLs for system modeling.

The prototypical implementation of LCA shows that the integration of abstract syntax of DSLs is straightforward, based on object-oriented metamodels. However, abstract syntax lacks operational semantics, which must also fit. But templates associated with metamodel elements might embody operational semantics based on a particular target technology. With *simCore* the presented metamodel defines the framework for the definition of further DSLs, which is illustrated in the following chapter at the example of *ECAL*. One basic characteristic of the approach presented here is that with the given metamodeling technologies, the development of DSLs roughly follows the "look and feel" of object-oriented development of modeling frameworks, insofar a general structure and semantics is prescribed by abstract classes (e.g. *SubModel*), which have to be specified at lower levels. The major distinction however is that instances to not present the software under development (i.e. simulator), but the specifications of models that in the case under consideration represent both simulator and real systems. Instead of programming with GPL, one has to specify code generation templates, which is programming at the meta-level. A major practical issue is that there is no feedback from the simulator to the model specification of the DSL, thus with the used technologies there is no runtime monitoring or debugging possible in the modeling tool, but only with the framework-level tools (i.e. Java development tools).

Algorithm 2 Simulation procedure for coupled models (pseudocode).

```
ParallelModels ▷ set of models that schedule times of transition (PARALLEL)
OnEventModels ▷ set of ONEVENT_SERIAL) coupled models
Simtime ← 0 ▷ simulation time set to 0
Events ← ∅ ▷ set tuples with time and model
for all model ∈ (ParallelModels ∪ OnEventModels) do
    model.Initialize() ▷ set to initial state
    model.OutputFunction() ▷ execute output function
end for
for all model ∈ ParallelModels do
    Events ← Events ∪ (0, model) ▷ schedule the model for transition at t_0
end for
moreEvents ← true
while moreEvents do ▷ loop all times of events until end of simulation
    DueParallelModels ▷ Set of models with scheduled transitions at Simtime
    for all model ∈ DueParallelModels do
        model.InputFunction() ▷ execute input function
    end for
    for all model ∈ DueParallelModels do
        nextTime ← model.TransitionFunction() ▷ execute transition from which
follows the time of the next event
        model.OutputFunction() ▷ execute output function
        Events ← Events ∪ (nextTime, model) ▷ Schedule next event
    end for
    for all model ∈ DueParallelModels do
        dueModels ▷ models that are coupled ONEVENT_SERIAL to model
        ExecuteOnEventChain(dueModels)
    end for
    Events ← RemoveEvents(Simtime) ▷ remove processed events
    if Events ≠ ∅ then
        Simtime ← minTime(Events) ▷ Set next Simtime to minimum scheduled time
    else
        moreEvents ← false
    end if
end while
function ExecuteOnEventChain(dueModels)
    for all model ∈ dueModels do
        model.InputFunction() ▷ execute input function
        model.TransitionFunction() ▷ execute transition
        model.OutputFunction() ▷ execute output function
    end for
    for all model in dueModels do
        dueModels ▷ models that coupled ONEVENT_SERIAL to model
        ExecuteOnEventChain(dueModels)
    end for
end function
```

6 The Environmental Cellular Automata Language (ECAL)

The Environmental Cellular Automata language (ECAL) is developed as modeling language for EMS that particularly aims at providing means to describe Cellular Automata as characterized in Chapter 4.4. One goal of design is to set the language into the tradition of micro-scale modeling physical processes. Based on the assumption that findings related to micro-scale modeling are a major source of intuition of modelers, this may support the transfer of ideas. A guiding principle is the avoidance of unwanted generality, thus intransparency, by inclusion of low-level language elements of GPL. Thus, ECAL is not designed to introduce some new feature to the notion of CA modeling, it is rather meant to be a representation of common aspects of CA as used in wide areas of EMS. This may support relatively explicit representations of mechanisms under consideration and support transfer of mechanisms between fields of investigation, while it aligns with common practices in EMS.

6.1 Basic Language Concepts

As a language that is based on an object-oriented meta-model, concrete syntax is rather a secondary property of the language definition. However, since the textual concrete syntax is realtively compact in many cases, in particular mathematical expressions, concrete syntax is mainly used for illustration. Further, since the the operational semantics is finally given in terms of a transformation into code of GPL (Java in case of the prototype presented here), the definition of operational semantics recurs to "common features" of typical GPL at the base of pseudo code. Since there is a variety of computer languages that fall under this notion and the perception of a computer language being "typical" and a feature being "common" being subjective, it must be noted that the semantics of ECAL has been actually defined tested based on a transformation into the Java programming language. Thus definitions finally refer to Java as the semantic base, although it aims at some generality with respect to "similar" programming languages.

According to the common notion of macro-scale CA (see 4.4), the evolution of the CA proceeds in fixed time steps, thus it is a discrete-time dynamical system and the lattice consists of a two-dimensional finite number of adjacent rectangular cells[1]. Further, update is perceived as synchronous in that at every timestep the same transition function is applied to the cells.

Further fundamental characteristics are the explicit consideration of:

- the geospatial reference,
- the notion of a cell, as opposed to matrizes, layers etc.,

[1] Although three-dimensional CA are common too, for practical purposes the investigations of this thesis relate to two dimensions.

- the notion of sets of cells for modeling (clusters, neighborhoods, candidates etc.) and
- the explicit ordering of different parts of the transition function.

At the global level, an ECAL specification consists of different distinct parts for the description of the most basic properties of the CA (identifier, extent, boundary condition and stepsize, l.1), the geospatial reference (l.2), global variables (l.3), the cells (l.4), initialization (l.5), transition function (l.6), input-output transformation (l.7) and output-input transformation (l.8).

```
1 Eca (name: "Model1" width: 100 height: 100  Boundary: CUT stepsize: 1 )
2     SpatialReference : ... //the spatial reference
3     GlobalStateVariables { ... } //global variables and parameters
4     CellDefinition { ... } //attributes of the cell
5     Initialisation{ ... } //initialization
6     EcaTransitions{ ... } //transition function
7     InputFrame{ ... } //input - state mapping
8     OutputFrame{ ... } //state - output mapping
9     InputPorts{ ... } //input ports
10    OutputPorts{ ... } //output ports
11 EndEca
```

The meaning of basic properties is straightforward: *name* specifies an identifier for the model as string, width and height are the number of cells in the two dimensions, where *width* corresponds to cells in East-West direction and *height* in North-South direction. *Boundary* specifies a predefined boundary condition. Available well-known boundary conditions are *CUT* (none), *MIRROR* (reflecting) and *WRAPPED* (cyclic)). The *stepsize* defines the increment of time per timestep.

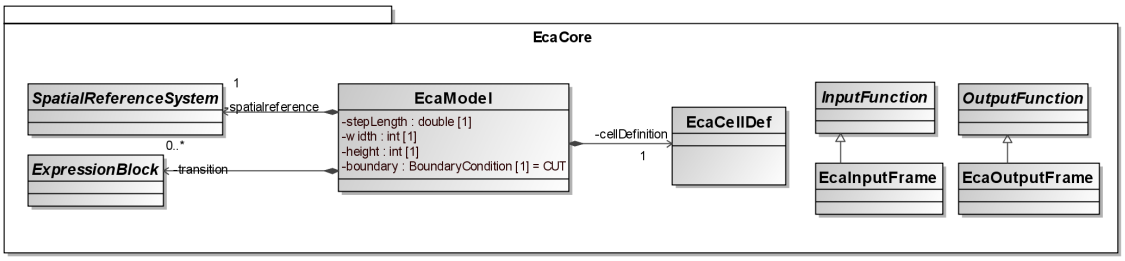

Figure 6.1: The kernel of the metamodel of ECAL.

Figure 6.1 presents the kernel of the package of the metamodel of ECAL. The basic element is *EcaModel* as a subclass of *simCore.SubModel*. *ECAInputFrame* and *ECAOutputFrame* are specifications of *InputFunction* and *OutputFunction*. The association *cellDefinition* corresponds to *CellDefinition* in concrete syntax and *ExpressionBlock* is a subclass of *simCore.TransitionFunction*.

The overall simulation procedure is prescribed by *simCore* as defined in Chapter 5.4.1 by Algorithm 2. If a transition is due, the execution proceeds as follows: first the directives of the *InputFrame* are executed as the ECAL-specialization of *simCore InputFunction*. The *InputFrame* defines a function that maps the values of input port variables - defined in *InputPorts* - to state variables that are defined in *GlobalStateVariables* and *CellDefinition*.

Second, the directives defined in *EcaTransitions* calculate new values of state variables. Third, *OutputFrame* defines a function that maps the values of state variables to values of output port variables, which are defined in *OutputPorts*. Finally, the next transition is scheduled to $Simtime + stepsize$.

Geospatial Reference Each ECAL model is associated with a specific portion of geospace. According to the common notion of geospatial reference (see Chapter 5.3) the basic geometric properties of are defined at the base of a geospatial reference system and the envelope (bounding box) that is the smallest rectangular geometry that includes all geometries of the lattice. There are some alternative ways to specify *SpatialReference*, i.e. using one of the spatial reference systems defined by EPSG, complemented by a specification of the envelope.

```
SpatialReference : Crs(CrsEPSG:"4326") Bounds(Up:52.0, Low:50.0, Left:10.0,
    Right:13.0);
```

By this, every cell is identified with a geospatial location, thus a coordinate in terms of the geospatial reference chosen, and a CA location, denotes the position of a cell as a tuple *(X, Y)* indicating the position counted from West to East (X) and the position counting from North to South (Y) relative to the North-West corner. Figure 6.2 illustrates the correspondence of the spatial reference in terms of cell position within the lattice and the geospatial reference given by means of an envelope that is defined at the base of a geospatial reference system.

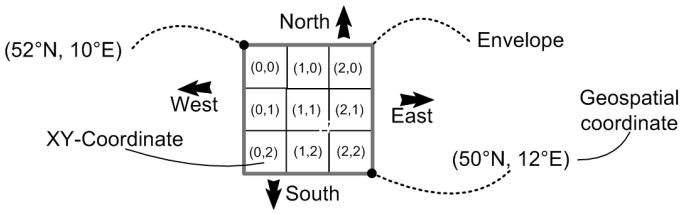

Figure 6.2: The spatial and geospatial reference of cells in ECAL.

Variables and Parameters The state of the CA is modeled at the base of variables and parameters. Variables are variant, thus their value can be changed in the transition, in contrast to parameters. Variables and parameters may be declared as global variables within the *GlobalVariables* block, where each variable is given a unique identifier and a type. Global variables are globally accessible in the sense of usual GPL such that their value can be read and set from anywhere in the specification of the transition function.

```
1    GlobalStateVariables {
2        State Parameter Real Param1 ;
3        State Parameter Int Param2 ;
4        State Variable List <Cell> Var3 ;
5    }
```

There are three types of variables and parameters for modeling the state: *Real* that corresponds to real numbers, *Int* that corresponds to the Integers and the type *List*. *List*-type variables reference a set of values of one of the basic types, which is given as a declaration parameter (within "<" and ">")[2]. Numeric types *Int* and *Real* correspond to the notion of common programming languages with respect to operations[3]. Variables and parameters must be declared with a type and may be initialized. If no initialisation value is given a default value is given that is *0.0* for type *Real*, *0* for *Int* and an empty list for *List*. The type *List* is an unordered set of (references to) elements, with duplicates allowed. Lists of type *Cell* are a central concept of ECAL and come along a characteristic set of operations (see below). The distinction between variables and parameters is made explicit, indicated by the keywords *parameter* and *variable*[4].

Cells Cells are characterized by their position in the lattice and the value of characterizing variables (and parameters[5]). Each cell has the same set of variables which are defined by the modeler, which is further referred to as "cellstate" and corresponding variables as "cellstate variable". A cellstate is defined by a series of cellstate-statments within a *CellDefinition* block:

```
CellDefinition
{
    Cellstate Variable List <Cell> Neighbors ;
    Cellstate Variable Int statevariable1;
    Cellstate Variable Real statevariable2;
}
```

Each numeric type cellstate corresponds to a layer in that results from a straightforward spatial composition of a cellstate as the union of all geometries of cells and corresponding variable values. Each cellstate of type *List* of type *Cell* corresponds to a neighborhood of a cell, since it contains a list of references to cells. There are various ways to define lists of Cells (see below). A cell itself can be imagined as a data structure, e.g. an object or structured type with variables that correspond to the cellstates. However, there are further cellstate variables not explicitly declared:

- X: the X-coordinate of the cell of type *Int*, which is invariant.
- Y: the Y-coordinate of the cell of type *Int*, which is invariant.
- *Transient cellstate variables*: for each declared cellstate variable a second variable of the same type is added

[2] At the time of writing this thesis only lists of type *Cell* are defined and implemented.

[3] In fact, *Int* and *Real* are mapped onto Java types *double* and *int* such that semantics of corresponding basic operations (+,-,<,>,==) is given by the corresponding Java semantics.

[4] In concrete syntax, identifiers are used for referencing models, variables and parameters. ECAL imposes the restriction on identifiers that they may be composed of letters, numbers, the underscore (_) only and may not start with a number and may not be keywords of the concrete syntax. One restriction is an artifact of the (prototypical status) of the tool support that is used to generate the model editor: identifiers must be unique in all contexts (e.g. no shadowing).

[5] This distinction is further only mentioned, when non-obvious differences between variables and parameters exist.

The corresponding Java class (Listing 6.1) illustrates the semantic foundation of cells:

Listing 6.1: A *Java* class corresponding to cell definition in *ECAL*

```
public class Cell {
    int X;
    int Y;
    int statevariable1Java; // state variable
    int statevariable1Trans; //transitory state variable
    double statevariable2Java;
    double statevariable2TransJava;
    Object [] NeighborsJava;
    Object [] NeighborsTransJava;
}
```

Variables of type *Cell* allow access to the values of corresponding variables and parameters:

- *Cellstate variable reference*: provides access to variable or parameter value (denoted by the symbol -> following *$* and the name of the cell variable, see below)
- *Assignments*: assignments of values to cellstate variables assign the new value to the corresponding transient cellstate variable.
- *Equality*: the equality operator == returns *true* when the values of the corresponding *X* and *Y* cellstate variables are both equal, *false* otherwise.

Transitions Generally computations can be specified in the blocks *Initialisation*, *Eca-Transitions*, *InputFrame* and *OutputFrame*. For modeling change of variable values, the basic directive is the assignment, denoted by =, that assigns a value specified at the right hand side to a variable at the left hand side. There are basically two ways of structuring the flow computations: *GlobalTransition* and *ForEachCell* blocks. The following snippet illustrates these within an *Initialisation* block.

```
Initialisation{
    GlobalTransition if (true) {
        statevariable1=1
        RepeatUntil{false}
    }
    ForEachCell In [AllCells] INSTANT {
        $this->Neighbors= Name=MOORE EXCLUSIVE
        if[X(this) >= 10] then [ $this->statevariable1 = 0 ]
    }
}
```

A *GlobalTransition* transition contains a number of directives that are executed in the order of specification. Execution may be conditioned by a conditional statement (*if()*) and may be repeated until stopping condition is satisfied (i.e. *RepeatUntil{}*). The *ForEach-Cell*-directive applies the enclosed calculations in the order of specification to all cells of a given set of cells (i.e. *AllCells*, assignment of the Moore-neighborhood to the cellstate variable *Neighbors* each cell). Assignments to cellstate variables are allowed only *ForEach-Cell* directives and in the *Initialisation* directive. *IfThen* statements specify conditional execution.

Within both types of blocks, variables might be introduced which then are accessible (read and write) within the blocks itself and all blocks nested inside. In combination with lists of cells, this can be used for the specification of dynamical hierarchical contexts:

```
1 GlobalTransition{
2   /*
3   Variable List <Cell> cluster;
4   ...
5   /* some calculations populating the List cluster */
6   ...
7   Variable Int count =
8     Count(Cells In [$cluster] Having [true];
9
10  ForEachCell In [cluster] AS source INSTANT {
11    ForEachCell In [$clusterCell->Neighbors] AS nb NONE {
12        if[$nb->Alive == 0 AND count < 10] then [$nb->Alive=1];
13    }
14  }
15 }
```

The members of a cluster are stored in a variable (i.e. *cluster*, l.2), some aggregation takes place (i.e. the number of cells in the cluster is retrieved) and stored in a variable of the context of the global transition function (i.e. *count*, ll.6/7). Two nested *ForEachCell* blocks describe spread from the members of the cluster (*source*) to neighbors (*target*) that is ultimately dependent on the size of the cluster, thus a dynamically changing property[6].

According to the basic notion of parallel update within classic CA, by default, the change of the value of a cellstate variable is not visible, until the next timestep (at the semantic level, values are assigned to the corresponding transitory variable). To change the default behavior the keyword *INSTANT* indicates, that the state change will be persisted after all cells of the *ForEachCell*-directive have been considered. *INSTANT_SERIAL* indicates that changes are persisted immediately (persisting is realized through assigning the value of the transitory variable to the corresponding state variable). *NONE* indicates default behavior.

In the example below, the transition in lines 1-3 would set the value of the cellstate variable *Neighbors* to 1, which is the state the following transition operates on (ll. 5-9). Thus, the update is parallel for all cells considered within a *ForEachCell* directive, but serial with respect to the subsequent directives. This enables a sequential ordering of parts of the transition function.

```
1 ForEachCell In [clusterCells] AS cell2 INSTANT {
2   $cell2->Alive = 1;
3 }
4
5 ForEachCell In [AllCells] AS cell NONE {
6   ForEachCell In [$this->Neighbors] AS nb INSTANT {
7     $nb->Alive=0;
8   }
9 }
```

However, the example above illustrates that when a *ForEachCell* statement is nested into another *ForEachCell* statement, there are concurrent updates on cells that belong to a neighborhood of more than one cell. A solution would be not to allow such nesting. However, such nesting naturally corresponds to CA, where spread processes occur probabilistically, where starting from a set of source cells (i.e. *AllCells*), spread processes are

[6] In reality such dependencies are typically more elaborated, see use cases below.

initiated probabilistically to target cells (i.e. to Neighbors). For this, the transition of *ForEachBlock* is executed serially in random order[7]. If the outer *ForEachCell* directive is *INSTANT* and the inner directive is *NONE*, all state changes that occurred within the scope of the inner directive are persisted according to the outer directive. Please note that in this case state changes of the inner directive might be lost, when updates are overwritten.

Algorithm 3 illustrates the simulation procedure of *ForEachCell* directives:

Selection and Aggregation The explicit selection of sets of cells and further processing is a central idea of ECAL. There are some possibilities for the selection of cells that retrieve Lists of type Cell

- *AllCells*: returns all cells.
- *SelectNeighborByName*: returns all cells of a predefined neighborhood of a cell (Moore or vonNeumann).
- *CellsWithinRadius*: returns all cells within a distance (geospatial units) measured as line between the center points of cells.
- *SelectCellsFrom*: returns a set of cells as defined by a selection condition.

The following examples correspond to cell selection expressions:

```
1 AllCells; //All cells
2 Name = MOORE EXCLUSIVE;
3 CellsWithinRadius ( 3.5 ) EXCLUSIVE;
4 SelectCells From {CellList} Having [$this->state == 0]];
5 SelectCells From {CellList} As cell Having [$cell->state == 0]];
```

SelectCells statements take as argument a list of type *Cell* and specify a condition (*Having[]*) that is a boolean expression that contains a reference to at least one of the cell's variables. Please note, that global and local variables might be referred within the condition.

Further, operations on lists of cells address the issue of expressing selections of cells:

- *UnionCellList*: set union with duplicates.
- *AppendCellList*: appends a list to another list.
- *RemoveCellFromList*: removes a cell from a list.
- *EmptyList* Removes all elements from a list.
- *Sort*, sorts the list according to an criterion (the criterion must be a boolean expression).
- *AddElement*, adds a cell reference to the list.

Other expressions derive aggregate characteristics of the cells within a list (Rank, Position, Count), e.g. the expression *Count(Cells In [$list] AS cellVar Having [$cellVar->state == 0]* counts the number of cells in the list of cells *list* with the having a value of variable *state* of *0*. The cell selection expression *Cells In [CELLSET] AS CELLREF Having*

[7] This indeed is a severe limitation to possibilities of parallelization.

Algorithm 3 Simulation procedure for ECAL ForEachCell directives (pseudocode)

```
Cells                                          ▷ the list of cells to which transitions apply
Cells ← RandomizeOrder(Cells)                  ▷ randomize the order of the cells
i ← 0
while i < size(Cells) do
    cell ← Element(Cells, i)                   ▷ consider ith cell of the list
    ...                                        ▷ Allocate and initialize local variables
    ...                                        ▷ Execute directives (assignments and nested blocks)
    i = i + 1
    if INSTANT_SERIAL then
        PersistState()                         ▷ persist state
    end if
end while
if INSTANT then
    PersistState()
end if
function RandomizeOrder(cellList)
    cellListRandom ← new List                  ▷ Create new list
    while empty(cellList) ≠ true do            ▷ loop while there are elements in cellList
        int idx = Random (0, size(cellList))   ▷ draw random number from all available
indizes
        Append(cellListRandom, Value(cellList,idx))   ▷ append the cell variable at
position idx to the end of the new list
        Remove(cellList, idx)                  ▷ the cell variable at position idx from source list
    end while
    Return cellListRandom
end function
function PersistState
    i ← 0
    AllCells                                   ▷ a list of all cells
    while i < size(AllCells) do                ▷ loop over all cells
        cell ← Element(AllCells, i)            ▷ consider ith cell of the list and assign to cell
        for all variable in cell do            ▷ Loop all declared cellstate variables
            cell->variable ← cell->variable_trans   ▷ assign value of the corresponding
transitory variable to the cellstate variable
        end for
        i = i + 1
    end while
end function
```

[COND] is a basic element of ECAL used for the specification of lists of cells at the base of cellstate and global and accessible local variables. The meaning is that for every element in CELLSET it is evaluated if the boolean condition COND is true. If true, the element is added to the resulting list of cells (Chapter 6.2 illustrates its application).

Listing 6.2: Port and frame definition in *ECAL*.

```
1 InputPorts{
2   Name = InPort1, DataType = GridCoverage <Int>;
3   Name = InPort2, DataType = Int;
4 }
5
6 OutputPorts{
7   Output { Name = OutPort1, DataType = Real};
8   Output { Name = OutPort2, DataType = GridCoverage <Int>};
9 }
10
11 InputFrame{
12   GlobalLevelCalculation{
13     cellstate1=ToCellstate(InputPort(Port1));
14     globalVar1=InputPort(InputPort2);
15   }
16 }
17
18 OutputFrame{
19   GlobalLevelCalculation{
20     OutputPort(OutPort1)=ToGridCoverage($this->cellstate1);
21     OutputPort(OutPort2)=globalVar1;
22   }
23 }
```

Listing 6.2 illustrates the definition of ports and input and output functions. Input ports (ll. 1-4) and output ports (ll. 6-9) are simply defined by giving an identifier and a datatype. Note that the datatype must be available in *simCore*. Lines 11-16 specify the input frame that maps input to state variables and lines 18-22 specify the output function that maps state to output ports. For being able to use external input as values for cellstate variables and making the value of cellstate variables available at ports, a conversion is defined (*ToGridCoverage*, l. 20) that transforms a cellstate to a corresponding grid coverage and *ToCellState* (l. 13) that sets values cells according to a grid coverage. The transformation is straightforward since the spatial reference of the CA model is identical to the spatial reference of the grid.

ECAL and LCA have been used to reimplement some CA macro-scale models. The most comprehensive study was a study of land-use change that is presented in the following Chapter 6.2 in order to illustrate ECAL and LCA at a "real-world" example. Appendix C presents the reimplementation of a model that combines fire spread and fire fighting activities. Appendix D presents the reimplementation of a model that models fire spread including a "spotting process" modeled by a spatially discountinous process.

6.2 Case study: Land Use Change Modeling with SLEUTH

SLEUTH is a well-established approach to model land use change and comprehensively documented in SLE (2012)[8].

[8]SLEUTH is an acronym for input datasets: slope, landuse, excluded, urban, transportation and hillshade.

6.2.1 General Setting of the Study

The overall aim of this case study was to apply one component of SLEUTH - the urban growth model (UGM) - to modeling urbanization in the region of Greater Tirana (Albania). However, this requires the adaption of the model to match the available data. Figure 6.3 illustrates the overall procedure of the case study. The original SLEUTH model is a cellular automaton model that takes as inputs image datasets that represent four spatial characteristics and calculates spatial patterns of land use (urbanization) from them:

- one image modeling the slope,
- at least four images that represent the urban land use at four times,
- at least one image that represents that transportation network and
- at least one image that represents those areas that are excluded from urbanization (e.g. lakes).

In the study the UGM has been reimplemented with ECAL and associated DSLs and then modified and the modified version has been compared with the reimplemented version of the original model. First results have been published in Theisselmann et al. (2010) and indicate a slight gain in performance[9].

At the base of inputs, the simulator executes a calibration routine, where the model is simulated for a range of parameter values and a number of Monte-Carlo repetitions with the random number generator initialized differently for each repetition. Different statistical measurements are calculated that provide measurement of simulated and observed urbanization. Since an exhaustive simulation of parameters is typically beyond possibilities, a procedure is suggested, where calibration proceeds from coarse grain variations of parameter values to finer granularity for subregions of parameter space that are selected by the modeler.

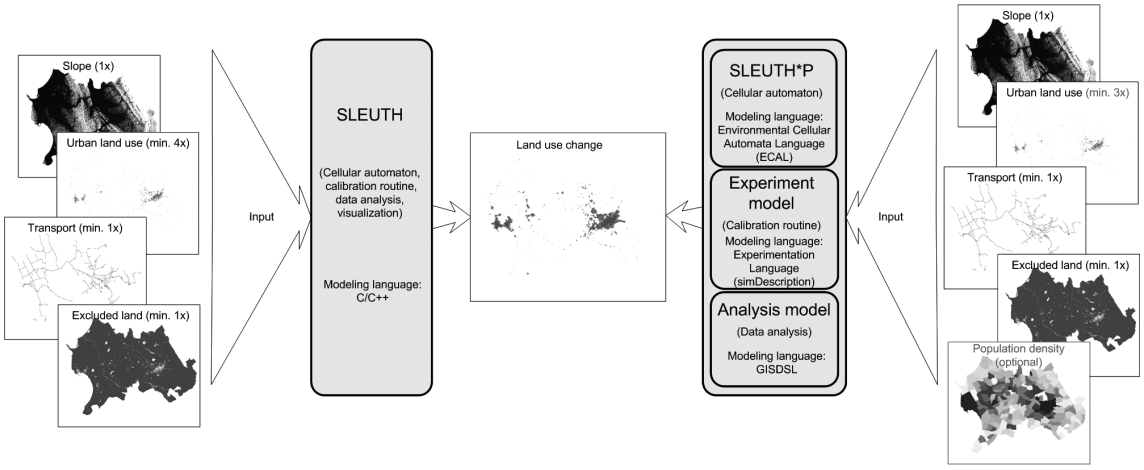

Figure 6.3: Illustration of the SLEUTH use case.

The application of the original SLEUTH software to the region of Greater Tirana is not possible, because the available data does not meet the requriements of the built-in

[9] A detailed description of results is submitted for publication at the time of writing this thesis.

calibration routine. Moreover, there is data on population density available for this region that cannot be processed by the original SLEUTH implementation, but it is feasible to assume that the exploitation of the information in this data might enhance accuracy of the SLEUTH approach. Thus, the study encompasses a modification to the state of the CA model, such that it encompasses the influence of population density and the modification of the calibration routine since not enough datasets are available for applying the original routine. The original SLEUTH implementation is based on the programming language C and intended to be used as "black-box" model, thus modifications are not considered, which motivated the use of ECAL.

The SLEUTH model contains two components, the *Urban Growth Model* (UGM) and the *Deltatron* component. In this study, only the UGM is considered and reimplemented, since Deltron is not relevant.

The UGM basically models 4 types of urban growth processes by means of transition rules:

- *Spontaneous growth* (SG): random urbanization.
- *New spreading center growth* (NSCG): growth at newly urbanized areas.
- *Edge growth* (EG): growth at the edge of spreading center.
- *Road influenced growth* (RIG): growth along the transportation network.

The different growth processes are modeled by means of corresponding probabilistic transition rules, which are invariant. However, five coefficients determine the absolute and relative influence of the different growth processes. The values of these coefficients that are specific to the study area are determined by executing calibration routine and are variable in that their values can change according to the evolution of the CA.

Generally, the transition rules are modeled at two levels: the local (cell) and the global (cellular automaton) level. At the local level, UGM computes a development potential for selected cells based on nearness to roads, topographic slope, and constraints on land to be urbanized with already developed cells being the seeds for growth. Although the description of the transition rules is rather complicated and exceeds simple "if-then-rules", SLEUTH basically relies on the idea that in a cellular space a series of transition rules are enforced to govern the state of cells depending on the local configuration of its neighborhood.

6.2.2 Implementation of the UGM using ECAL

This section presents the reimplementation of the original UGM model with ECAL and the modifications applied at the base of ECAL source code. The changes made for the modified version are highlighted with gray color. The specifcation of an experiment with *simDescription* and the data analysis with *GISDSL* are provided in Appendix B.

Listing 6.3 presents the declaration of global variables and parameters: along the five variable coefficients (ll. 3- 7), there are 18 parameters taken from the original model (ll. 8 - 25), a cell list variable for all cells but the boundary cells (l. 29) and a variable for storing the growth rate of the urban land use (l. 30). Line 8 introduces a new coefficient that models the influence of population density analogously to the original coefficients. Lines 27 and 28 introduce further parameters that are used in edge growth modeling.

Listing 6.3: The global state variables of UGM in *ECAL*.

```
1   GlobalStateVariables
2   {
3       State Variable Real DispersionCoeff ; <"Dispersion/diffusion
            coefficient">
4       State Variable Real BreedCoeff ; <"Breed coefficient">
5       State Variable Real SpreadCoeff ; <"Spread coefficient">
6       State Variable Real SlopeCoeff ; <"Slope coefficient">
7       State Variable Real RoadCoeff ; <"Road coefficient">
8       State Variable Real PopCoeff ; <"Population density coefficient">
9       State Parameter Real CRITICAL_HIGH ; <"Critical upper growth rate
            ">
10      State Parameter Real CRITICAL_LOW ; <"Critical lower growth rate
            ">
11      State Parameter Real BOOM ; <"Coefficient modifaction factor in
            boom phase">
12      State Parameter Real BUST ; <"Coefficient modification in bust
            phase">
13      State Parameter Real SlopeSensitivity ;
14      State Parameter Real RoadSensitivity ;
15      State Parameter Real MIN_SLOPE_RESISTANCE ;
16      State Parameter Real MIN_ROAD_GRAVITY ;
17      State Parameter Real MIN_DIFFUSION ;
18      State Parameter Real MIN_SPREAD ;
19      State Parameter Real MIN_BREED ;
20      State Parameter Real MAX_SLOPE_RESISTANCE ;
21      State Parameter Real MAX_ROAD_GRAVITY ;
22      State Parameter Real MAX_DIFFUSION ;
23      State Parameter Real MAX_SPREAD ;
24      State Parameter Real MAX_BREED ;
25      State Parameter Real CRITICAL_SLOPE ;
26      State Parameter Real MAX_ROAD_VALUE ;
27      State Parameter Real CRITICAL_MAX_POP ;
28      State Parameter Real CRITICAL_MIN_POP ;
29      State Parameter List <Cell> InnerCells ;
30      State Variable Real GrowthRateVar ;
31  }
```

Listing 6.4 presents the declaration of the cell state variables, which are the invariant parameters corresponding to the input datasets slope, land use and excluded (ll. 3 - 5), two variables corresponding to input datasets urban and transportation (ll. 6/7) and a variable for the neighborhood (l. 9). Line 8 introduces a cellstate variable that corresponds to the added population density input.

Listing 6.4: The cell state variables and parameters of UGM in *ECAL*.

```
1   CellDefinition
2   {
3       Cellstate Parameter Real Slope ;
4       Cellstate Parameter Int Landuse ;
5       Cellstate Parameter Int Excluded ;
6       Cellstate Variable Int Urban;
7       Cellstate Variable Int Transportation ;
8       Cellstate Variable Int Popdens ;
9       Cellstate Variable List <Cell> Neighbors ;
10  }
```

Listing 6.5 presents the initialisation specification, which simply states that cellstate *urban* is set to 0 and ech cell is assigned the Moore neighborhood (ll. 2-5) and that the values of variables and parameters except *InnerCells* are described in the experiment description indicated by the keyword *SetExternally* (ll. 6 - 36). *InnerCells* is set by a cell selection expression that selects all but the boundary cells (l. 37).

Listing 6.5: The initialization of UGM in *ECAL*.

```
1   Initialisation{
2       ForEachCell In [AllCells] INSTANT{
3           $this->Landuse=0;
4           $this->Neighbors=Name = MOORE EXCLUSIVE
5       }
6       GlobalTransition{
7           Slope=SetExternally;
8           Excluded=SetExternally;
9           Urban=SetExternally;
10          Transportation=SetExternally;
11          Popdens=SetExternally;
12          CRITICAL_HIGH=SetExternally;
13          CRITICAL_LOW=SetExternally;
14          DispersionCoeff=SetExternally;
15          BreedCoeff=SetExternally;
16          SpreadCoeff=SetExternally;
17          SlopeCoeff=SetExternally;
18          RoadCoeff=SetExternally;
19          BOOM=SetExternally;
20          BUST=SetExternally;
21          SlopeSensitivity=SetExternally;
22          RoadSensitivity=SetExternally;
23          MIN_SLOPE_RESISTANCE=SetExternally;
24          MIN_ROAD_GRAVITY=SetExternally;
25          MIN_DIFFUSION=SetExternally;
26          MIN_SPREAD=SetExternally;
27          MIN_BREED=SetExternally;
28          MAX_SLOPE_RESISTANCE=SetExternally;
29          MAX_ROAD_GRAVITY=SetExternally;
30          MAX_DIFFUSION=SetExternally;
31          MAX_SPREAD=SetExternally;
32          MAX_BREED=SetExternally;
33          CRITICAL_MAX_POP=SetExternally;
34          CRITICAL_MIN_POP=SetExternally;
35          CRITICAL_SLOPE=SetExternally;
36          MAX_ROAD_VALUE=SetExternally;
37          InnerCells=SelectCells From {AllCells} Having [X(this) > 0
                AND X(this) < EcaWidth AND Y(this) > 0 AND Y(this) <
                EcaHeight]
38      }
39  }
```

Listing 6.6 presents the skeleton of the the transition function of the UGM model as a global transition with a number of local variables (ll. 4 -21) that are used to store temporary values that are calculated during the transition defined within the rules that model the different urbanization processes. Whereas spontaneous growth and edge growth are nested (ll. 23/24) these are performed in series with the rules for organic growth (ll. 26/27), road influenced growth (ll. 29/30) and the rule for coefficient modification.

Listing 6.6: The transition function of UGM in *ECAL*.

```
1  EcaTransitions{
2      <"UGM transition function">
3      GlobalTransition{
4          Variable Int NumUrbanizedBefore ;<"Number of urbanized cells">
5          Variable Real DiffVal ;<"Number of cells that are selected for
               combined sponatenous and edge growth">
6          Variable List <Cell> NewlyUrbanizedCell ;<"Cells that are
               urbanized during the transition">
7          Variable Real SlopeWeight ;<"Factor that influences urbanization
               by a cell's slope">
8          Variable Real PopWeight ;<"Factor that influences urbanization by
                a cell's population density">
9
10         Variable Real Val ;<"Auxiliary to calculate slope weight">
11         Variable Real exp ;<"Auxiliary to calculate slope weight">
12         NumSpontGrowth=0;
13         NumEdgeGrowth=0;
14         NumOrganicGrowth=0;
15         NumRoadGrowth=0;
16         <"DiffVal/Number of Sponatnaeous growth tries: maximum 5perc of
               CA diagonal">
17         DiffVal=DispersionCoeff * 0.0050 * Sqrt(EcaHeight * EcaHeight +
               EcaWidth * EcaWidth );
18         exp=SlopeCoeff / (MAX_SLOPE_RESISTANCE / 2.0);
19         <"Empty list of urbanized cells for this year">
20         NewlyUrbanizedCell=NewList (Cell);
21         NumUrbanizedBefore=Count( Cells In [AllCells]   Having [$this->
               Urban > 0] );
22
23         <"Spontaneous growth & edge growth">
24         GlobalTransition{ ... };
25
26         <"Organic growth">
27         GlobalTransition{ ... };
28
29         <"Road influenced growth">
30         GlobalTransition if (...){ ... };
31
32        <"Coefficient modification">
33        GlobalTransition{ ... }
34     }
35 }
```

The growth rules are more interesting since they highlight typical characteristics of macro-scale CA models. Listing 6.7 presents the transition that describes spontaneous and edge growth. Spontaneous growth randomly selects a cell from all cells (ll. 7 - 13) and then probabilistically tries to urbanize this cell. The probability of urbanization is dependent on the global parameter *CRITICAL_SLOPE*, the slope, the value of excluded, the position and landuse (ll. 18-24), which is typical for the UGM model. All transitions follow this pattern. This simple modification reflects the observation that the attractivity rises with urbanization, thus population density. If a cell is urbanized, the state is changed accordingly (l. 22) and the cell is added to the lists *NewlyUrbanizedCell* (l.24) and *spreadEdge*. The modification introduced is also typical and can be found several times in the model: a weight (*PopWeight*) is calculated as a function of parameters and a local cell-state value (*Popdens*, l. 21), which affects the probability of urbanization (l.23). This is

repeated *DiffVal* times, which depends on the size of the CA and a global parameter (see Listing 6.6, l. 17).

Listing 6.7: Spontaneous growth and edge growth rules in *ECAL*.

```
1  <"Spontaneous growth & edge growth">
2  GlobalTransition{
3      Variable List <Cell> spreadSpont ;
4      Variable List <Cell> spreadEdge ;
5
6      GlobalTransition {
7          GlobalTransition{
8              <"Clear list">
9              spreadSpont=RemoveAllElementsFrom(spreadSpont);
10             <"Select cell randomly">
11             spreadSpont=AddElement [SelectRandomCell(AllCells)] To [
                   spreadSpont];
12             DiffVal=DiffVal - 1
13         }
14
15         GlobalTransition{
16           <"Urbanization">
17           ForEachCell In [spreadSpont] NONE {
18               Val=(CRITICAL_SLOPE - $this->Slope) / CRITICAL_SLOPE;
19               if[$this->Slope > CRITICAL_SLOPE] then [SlopeWeight=1.0];
20               if[$this->Slope <= CRITICAL_SLOPE] then [SlopeWeight=1.0 -
                     Pow(Val,exp)];
21               PopWeight=Min (($this->Popdens - CRITICAL_MIN_POP) / (
                     CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff / 100)
                     , 1);
22               if[X(this) > 0 AND X(this) < EcaWidth AND Y(this) > 0 AND Y
                     (this) < EcaHeight AND $this->Urban == 0 AND RandomReal
                     (0, 1) > SlopeWeight AND $this->Excluded <
                     RandomIntUniform(0, 100)
23                     AND RandomReal(0, 1) > SlopeWeight - PopWeight
24               ]
25               then[$this->Urban=2;
26                 NumSpontGrowth=NumSpontGrowth + 1;
27                 NewlyUrbanizedCell=AddElement [this] To [
                       NewlyUrbanizedCell];
28                 spreadEdge=AddElement [this] To [spreadEdge]]
29           }
30         }
31     RepeatUntil {DiffVal < 0}
32     };
33
34     <"Edge growth">
35     GlobalTransition{
36       Variable List <Cell> breedCells ;
37       breedCells=NewList (Cell);
38
39       <"Select breeding cells based on breed coefficient">
40       ForEachCell In [spreadEdge] NONE{
41         PopWeight=Min ((($this->Popdens - CRITICAL_MIN_POP) / (
               CRITICAL_MAX_POP - CRITICAL_MIN_POP) * PopCoeff, 100);
42         if[RandomIntUniform(0, 101) < BreedCoeff
43                                           +PopWeight]
```

```
44          then[breedCells=AddElement [this] To [breedCells]]
45        };
46
47        ForEachCell In [breedCells] AS src NONE{
48          Variable Int countUrbanizedCells ;
49          Variable Int edgeSpreadTries ;
50          Variable List <Cell> breedNeighbors ;
51          Variable Real spreadPopFactor ;
52          Variable Real MAX_EDGE_GROWTH_TRIES ;
53          Variable Real MAX_EDGE_GROWTH ;
54          Variable Real MIN_EDGE_GROWTH_TRIES_LIMIT ;
55          Variable Real MIN_EDGE_GROWTH_LIMIT ;
56          Variable Real MAX_EDGE_GROWTH_TRIES_LIMIT ;
57          Variable Real MAX_EDGE_GROWTH_LIMIT ;
58
59          edgeSpreadTries=0;
60          countUrbanizedCells=0;
61
62
63          spreadPopFactor=Min (($this->Popdens - CRITICAL_MIN_POP) / (
                CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff / 100), 1);
64          MIN_EDGE_GROWTH_TRIES_LIMIT=8;
65          MIN_EDGE_GROWTH_LIMIT=2;
66          MAX_EDGE_GROWTH_LIMIT=20;
67          MAX_EDGE_GROWTH_TRIES_LIMIT=80;
68          MAX_EDGE_GROWTH_TRIES=MIN_EDGE_GROWTH_TRIES_LIMIT +(
                MAX_EDGE_GROWTH_TRIES_LIMIT - MIN_EDGE_GROWTH_TRIES_LIMIT) *
                spreadPopFactor;
69          MAX_EDGE_GROWTH=MIN_EDGE_GROWTH_LIMIT + (MAX_EDGE_GROWTH_LIMIT -
                MIN_EDGE_GROWTH_LIMIT) * spreadPopFactor;
70
71
72          GlobalTransition{
73            breedNeighbors=AddElement [SelectRandomCell($src->Neighbors)]
                  To [NewList (Cell)];
74
75            ForEachCell In [breedNeighbors] AS recipient INSTANT{
76              Val=(CRITICAL_SLOPE - $recipient->Slope) / CRITICAL_SLOPE;
77              if[$recipient->Slope > CRITICAL_SLOPE]
78              then[SlopeWeight=1.0];
79              if[$recipient->Slope <= CRITICAL_SLOPE]
80              then[SlopeWeight=1.0 - Pow(Val , exp)];
81              PopWeight=Min (($this->Popdens - CRITICAL_MIN_POP) / (
                    CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff / 100),
                    1);
82              if[
83              PopWeight -
84              SlopeWeight < RandomReal(0,1) AND $recipient->Urban == 0 AND
                    $recipient->Excluded < RandomIntUniform(0, 100)]
85              then[$recipient->Urban=3;
86                NumEdgeGrowth=NumEdgeGrowth + 1;
87                countUrbanizedCells=countUrbanizedCells + 1;
```

```
88                  NewlyUrbanizedCell=AddElement [recipient] To [
                        NewlyUrbanizedCell]]
89              };
90              edgeSpreadTries=edgeSpreadTries + 1
91              RepeatUntil{countUrbanizedCells >= MAX_EDGE_GROWTH OR
                    edgeSpreadTries >= MAX_EDGE_GROWTH_TRIES}
92              <" In the original MAX_EDGE_GROWTH is 2 and
                    MAX_EDGE_GROWTH_TRIES is 8 ">
93            }
94          }
95      }
96  };
```

Cells that have been urbanized by spontaneous growth are a potential source of edge growth and added to a corresponding list of cells (*spreadEdge*, l. 28) and to *NewlyUrbanizedCell* that records all urbanized cells. The edge growth rule (ll. 36 - 96) selects probabilistically cells from *spreadEdge*, where the probability depends on the breed coefficient (ll. 40 - 45). The selected cells are breed cells and put a corresponding list (*breedCells*, l. 44). Then, for each breed cell, a neighbor is selected randomly (l. 73) and it is tried urbanize the cell probabilistically (ll. 75-89). If urbanized, corresponding counts are incremented and the cell added to the list of urbanized cells (ll. 85-88). For each breed cell the rule is repeated maximally *MAX_EDGE_GROWTH_TRIES* times or it is stopped if *MAX_EDGE_GROWTH* neighbors have been urbanized (l. 91). In the unmodified UGM *MAX_EDGE_GROWTH_TRIES* was set constant to 8 and *MAX_EDGE_GROWTH* to 2. The modification it aims at rising the probability of edge growth with rising population density, reflecting the attractivity of urban areas with high population density.

The organic growth rule (see Listing 6.8) randomly selects a number of cells from all cells not at the boarder and which are not urbanized (l. 4), depending on the spread coefficient. If for each of the selected "organic growth cells" there are less than eight and more than 1 neighbors urbanized, then a neighboring cell is randomly selected (l. 12) and it is probabilistically tried to urbanize the cell (ll. 15-27). The modifications (l- 18 and l. 22) are analog to the previous ones.

Listing 6.8: Organic growth rule of UGM in *ECAL*.

```
1  <"Organic growth">
2  GlobalTransition{
3    Variable List <Cell> OrganicGrowthCells ;
4    OrganicGrowthCells=SelectCells From {InnerCells}  Having [$this->Urban
         != 0 AND RandomIntUniform(0, 101) < SpreadCoeff];
5
6    ForEachCell In [OrganicGrowthCells] AS osrc NONE{
7      Variable List <Cell> orgGrowthTarget ;
8      Variable Int urbanCount ;
9      urbanCount=Count( Cells In [$osrc->Neighbors]   Having [$this->Urban
           > 0] );
10     orgGrowthTarget=NewList (Cell);
11     if[urbanCount >= 2 AND urbanCount < 8]
12     then[orgGrowthTarget=AddElement [SelectRandomCell($this->Neighbors)]
           To [orgGrowthTarget];
13
14       ForEachCell In [orgGrowthTarget] AS trgt INSTANT{
15         Val=(CRITICAL_SLOPE - $trgt->Slope) / CRITICAL_SLOPE;
16         if[$trgt->Slope > CRITICAL_SLOPE]
```

```
17          then[SlopeWeight=1.0];
18          PopWeight=Min (($this->Popdens - CRITICAL_MIN_POP) / (
                CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff / 100), 1);
19          if[$trgt->Slope <= CRITICAL_SLOPE]
20          then[SlopeWeight=1.0 - Pow(Val , exp)];
21          if[$trgt->Urban == 0  AND $trgt->Excluded < RandomIntUniform(0,
                100) AND RandomReal(0, 1) > SlopeWeight
22                          - PopWeight
23          ]
24          then[$trgt->Urban=4;
25            NumOrganicGrowth=NumOrganicGrowth + 1;
26            NewlyUrbanizedCell=AddElement [SelectCell Having [dX=0, dY=0]]
                  To [NewlyUrbanizedCell]]
27        }
28      ]
29    }
30  };
```

Listing 6.9 presents the road growth rule which is basically a random walk that is conditioned by the road network data. From the formerly urbanized cells (*NewlyUrbanizedCell*) a number of cells is randomly selected, with the number depending on the value of the breed coefficient (ll. 12-14). The selected cells are the source for random walk along the road. For this, cells with a road present are selected within a given search radius, where the the distance is manhattan distance (*CellsByHopDistance*, ll. 34-36). If no road cell is found, the procedure is repeated with an enlarged search radius until a maximum radius is reached (*RoadGravity*, l. 37). If road cells have been found, one is selected randomly (ll. 42/43) and from the road cells within neighborhood of this cell (*roadCells*, l. 44) again one is selected randomly. From this cell a random walk is performed along the road by iteratively selecting neighboring cells (l. 52) probabilistically, until a condition is met or the end of the road reached (ll. 47 - 56).

Listing 6.9: Road influenced growth rule of UGM in *ECAL*.

```
1  <"Road influenced growth">
2  GlobalTransition if (Count( Cells In [NewlyUrbanizedCell]    Having [true]
       ) > 0){
3    Variable Real RoadGravity ;
4    Variable List <Cell> roadSpreadingCells ;
5    Variable Int RoadCells ;
6    RoadCells=0;
7    RoadGravity=Cut(RoadCoeff / MAX_ROAD_GRAVITY * ((EcaHeight + EcaWidth)
         / 16.0),0);
8    roadSpreadingCells=NewList (Cell);
9
10   <"Select cells that are source for road walks">
11   GlobalTransition{
12     roadSpreadingCells=AddElement [SelectRandomCell(NewlyUrbanizedCell)]
           To [roadSpreadingCells];
13     RoadCells=RoadCells + 1
14     RepeatUntil{RoadCells >= BreedCoeff + 1}
15   };
16
17   <"Road influenced growth">
18   ForEachCell In [roadSpreadingCells] AS rssrc NONE{
19     Variable List <Cell> roadCells ;
20     Variable Cell roadCell ;
```

```
21      Variable Real run ;
22      Variable Real RunValue ;
23      Variable Cell roadNeighborCell ;
24      Variable List <Cell> RoadNeighborCells ;
25      Variable Int SearchRadius ;
26      RoadNeighborCells=NewList (Cell);
27      roadCells=NewList (Cell);
28      run=0;
29      SearchRadius=1;
30
31      <"Look for a road near the source">
32      GlobalTransition{
33        roadCells=NewList (Cell);
34        roadCells=CellsByHopDistance (rssrc,SearchRadius, EXCLUSIVE);
35        roadCells=SelectCells From {roadCells} Having [$this->
              Transportation > 0];
36        SearchRadius=SearchRadius + 1
37       RepeatUntil{SearchRadius > RoadGravity OR Count( Cells In [roadCells
             ] Having [true] ) > 0}
38      };
39
40      <"Select a road cell">
41      GlobalTransition{
42        if[Count( Cells In [roadCells] Having [true] ) > 0]
43        then[roadCell=SelectRandomCell(roadCells);
44             roadCells=SelectCells From {$roadCell->Neighbors} Having [$
                 this->Transportation > 0]]};
45
46             <"Walk along the road">
47             GlobalTransition{
48               if[Count( Cells In [roadCells] Having [true] ) > 0]
49               then[roadCell=SelectRandomCell(roadCells);
50                    run=run + 1;
51                    RunValue=$roadCell->Transportation / MAX_ROAD_VALUE *
                         DispersionCoeff;
52                    roadCells=SelectCells From {$roadCell->Neighbors}
                         Having [$this->Transportation > 0]];
53                    if[Count( Cells In [roadCells] Having [true] ) == 0]
54                      then[run=RunValue+1]
55             RepeatUntil{run > RunValue}
56             };
57             if[roadCell != NullCell]
58             then[roadNeighborCell=SelectRandomCell($roadCell->Neighbors);
59               RoadNeighborCells=AddElement [roadNeighborCell] To [
                     RoadNeighborCells]];
60
61             <"Urbanize cell near the road">
62             ForEachCell In [RoadNeighborCells] AS rNbr NONE{
63               Variable List <Cell> secRoadNeighbors ;
64               Variable Int c ;
65               Val=(CRITICAL_SLOPE - $rNbr->Slope) / CRITICAL_SLOPE;
66               if[$rNbr->Slope > CRITICAL_SLOPE]then[SlopeWeight=1.0];
67               if[$rNbr->Slope <= CRITICAL_SLOPE]then[SlopeWeight=1.0 - Pow
                     (Val , exp)];
68               if[$rNbr->Slope > CRITICAL_SLOPE]then[SlopeWeight=1.0];
69               c=0;
70               secRoadNeighbors=NewList (Cell);
71               PopWeight=Min (($this->Popdens - CRITICAL_MIN_POP) / (
```

```
                CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff / 100),
                 1);
72              if[X(rNbr) > 0 AND X(rNbr) < EcaWidth AND Y(rNbr) > 0 AND Y(
                    rNbr) < EcaHeight AND $rNbr->Urban == 0  AND $rNbr->
                    Excluded < RandomIntUniform(0, 100) AND RandomReal(0, 1)
                    > (SlopeWeight
73                                  - PopWeight
74                              )]
75
76              then[$rNbr->Urban=4;
77                   NumRoadGrowth=NumRoadGrowth + 1;
78                   NewlyUrbanizedCell=AddElement [rNbr] To [
                         NewlyUrbanizedCell];
79
80                   <"Select three neighboring cells">
81                   GlobalTransition{
82                    secRoadNeighbors=AddElement [SelectRandomCell($rNbr->
                         Neighbors)] To [secRoadNeighbors];
83                    c=c + 1
84                   RepeatUntil{c >= 3}
85                   };
86
87                   <"Urbanize neighboring cells">
88                   ForEachCell In [secRoadNeighbors] AS srNbr NONE{
89                    Val=(CRITICAL_SLOPE - $srNbr->Slope) / CRITICAL_SLOPE;
90                    if[$srNbr->Slope > CRITICAL_SLOPE] then[SlopeWeight
                         =1.0];
91                    if[$srNbr->Slope <= CRITICAL_SLOPE] then[SlopeWeight
                         =1.0 - Pow(Val , exp)];
92                    PopWeight=Min (($this->Popdens - CRITICAL_MIN_POP) / (
                         CRITICAL_MAX_POP - CRITICAL_MIN_POP) * (PopCoeff /
                         100), 1);
93                    if[X(srNbr) > 0 AND X(srNbr) < EcaWidth AND Y(srNbr) >
                         0 AND Y(srNbr) < EcaHeight AND $srNbr->Urban == 0
                         AND $srNbr->Excluded < RandomIntUniform(0, 100)AND
                         RandomReal(0, 1) > SlopeWeight
94                                      - PopWeight
95                    ]
96                    then[$srNbr->Urban=5;
97                         NewlyUrbanizedCell=AddElement [srNbr] To [
                              NewlyUrbanizedCell]]
98                  }
99              ]
100           }
101         }
102 };
```

From this road cell, one of the neighboring cells is selected randomly (ll. 57-59) and it is tested probabilistically for urbanization (ll. 63-75). If urbanized, three times one of the neighboring cells is selected randomly (ll. 81- 85), each of which is probabilistically tried to urbanize (ll. 87-98). Listing 6.10 presents the parameter modification that manipulates the coefficients such that if the growth rate, the ratio of newly urbanized cells and urbanized cells, exceed a value (*CRITCAL_HIGH*, l. 20), growth is accelerated (ll. 21-35) and if growth rate is below *CRITICAL_LOW* (l. 40), then growth is slowed down (ll. 41-52).

Listing 6.10: Coefficient modification rule of UGM in *ECAL*.

```
1  <"Coefficient modification">
2  GlobalTransition{
3    Variable Real RoadPixelCount ;
4    Variable Real TotalPixels ;
5    Variable Real ExcludedPixelCount ;
6    Variable Real Pop ;
7    Variable Real GrowthRate ;
8    Variable Real NewlyUrbanized ;
9    Variable Real PercentUrban ;
10   TotalPixels=EcaHeight * EcaWidth;
11   ExcludedPixelCount=Count( Cells In [AllCells]   Having [$this->Excluded
         > 0] );
12   Pop=Count( Cells In [AllCells]   Having [$this->Urban > 0] ) + Count(
         Cells In [NewlyUrbanizedCell]   Having [true] );
13   RoadPixelCount=Count( Cells In [AllCells]   Having [$this->
         Transportation == 1] );
14   NewlyUrbanized=Pop - NumUrbanizedBefore;
15   GrowthRate=NewlyUrbanized / Pop * 100;
16   GrowthRateVar=GrowthRate;
17   PercentUrban=(Pop + RoadPixelCount) / (TotalPixels - RoadPixelCount -
         ExcludedPixelCount) * 100;
18
19   <"Boom phase coefficient modification">
20   GlobalTransition if (GrowthRate > CRITICAL_HIGH){
21     SlopeCoeff=SlopeCoeff - PercentUrban * SlopeSensitivity;
22     if[SlopeCoeff <= MIN_SLOPE_RESISTANCE]then[SlopeCoeff=1.0];
23     RoadCoeff=RoadCoeff + PercentUrban * RoadSensitivity;
24     if[RoadCoeff > MAX_ROAD_GRAVITY]then[RoadCoeff=MAX_ROAD_GRAVITY];
25     if[DispersionCoeff < MAX_DIFFUSION]then[
26       DispersionCoeff=DispersionCoeff * BOOM;
27       if[DispersionCoeff > MAX_DIFFUSION]
28       then[
29         DispersionCoeff=MAX_DIFFUSION];
30       BreedCoeff=BreedCoeff * BOOM;
31       if[BreedCoeff > MAX_BREED]
32       then[BreedCoeff=MAX_BREED];
33       SpreadCoeff=SpreadCoeff * BOOM;
34       if[SpreadCoeff > MAX_SPREAD]
35       then[SpreadCoeff=MAX_SPREAD]
36     ]
37   };
38
39   <"Bust phase coefficient modification">
40   GlobalTransition if (GrowthRate < CRITICAL_LOW){
41     SlopeCoeff=SlopeCoeff + PercentUrban * SlopeSensitivity;
42     if[SlopeCoeff > MAX_SLOPE_RESISTANCE]
43     then[SlopeCoeff=MAX_SLOPE_RESISTANCE];
44     RoadCoeff=RoadCoeff - PercentUrban * RoadSensitivity;
45     if[RoadCoeff <= MIN_ROAD_GRAVITY]then[RoadCoeff=1.0];
46     if[DispersionCoeff > 0] then [
47       DispersionCoeff=DispersionCoeff * BUST;
48       if[DispersionCoeff <= MIN_DIFFUSION]then[DispersionCoeff=1.0];
49       SpreadCoeff=SpreadCoeff * BUST;
50       if[SpreadCoeff <= MIN_SPREAD] then [SpreadCoeff=1.0];
51       BreedCoeff=BreedCoeff * BUST;
52       if[BreedCoeff <= MIN_BREED] then [BreedCoeff=1.0]
53     ]
```

```
54     }
55 }
```

The description of one of the automated calibration experiment series conducted in this case study is displayed in Appendix B.

6.3 Conclusions

As intended, ECAL is a DSL for the description of CA models that provides degrees of freedom that are aligned with macro-scale modeling in the tradition of micro-scale physical modeling, but which is not based on GPL programming language constructs for transition modeling. It allows the explicit specification of synchronous update, serial update of parts of the transition function and serial update of single cells. The specification of asynchronous update is supported through the identification of sets of cells with corresponding transitions with possibly serial update of cells. Emergent higher-level geometries are supported insofar as it is possible to represent these as lists of cells for which means modification and analysis are supplied. However, explicit asynchronous scheduling of transitions is not supported. Spatial and temporal inhomogeneity is directly supported by means of explicit consideration and selection of sets of cells based on time and location, but basic geometries are invariant. The explicit consideration of lattices that correspond to single processes is not supported, however, if processes can be identified that execute transitions in series, these can be modeled as two coupled CA.

The application of ECAL shows that it can be applied to a real-world case study including the modification of a model. The case study highlights some general aspects of CA based modeling in that it is used to investigate assumptions - here assumptions about population density and urbanization - and directly test these assumption in an explorative approach by directly incorporating an assumed mechanism in the model. Explicitness of representation is supported inasmuch state changes are explicitly stated based on a clear semantic instead of application of nested functions that work on globally accessible arrays in the original C-based implementation. Further, the separation of the model of experiment, analysis and model may be considered supporting explicitness, thus transparency.

However, the use case also demonstrates that there are related issues: the specification contains repetitions, e.g. the probabilistic urbanization of cells, which could be resolved by reusable functions, which explicitly have not been considered for the sake of simplicity and explicitness of representation[10]. Further, the limitation of cellstate changes to be possible only within *ForEachCell* directives causes rather voluminous specification of transitions, where one cell is considered after another based on neighborhood (e.g. random walk). Thus from a pragmatic point of view *ECAL* has limits with respect to the size of specifications, when explicitness is an issue, in particular when repetitions are present. Although the inherent randomness of nested transitions at the cell level might reflect the style of modeling and may be perceived as more explicit than in GPL implementation, it is introduced implicitly, by means of the nesting structure and update scheme, although it presents a fundamental characteristic with respect to interpreting simulation results. Thus through the inherent probabilism, *ECAL* has pragmatic limitation with respect to nesting depth.

The evaluation of a computer language with respect to pragmatics in terms of regularity is generally difficult, since the assessment is always dependent on assumptions about

[10]Please note that, for the same reason, object-orientation has not been included, too.

the cognitive and educational status of users (see Chapters 3.5.3 and 3.2.3). However some remarks on regularity can be stated. ECAL is simple insofar it avoids generality of GPL, thus limits the possibilities of expressing the same computations, or rather makes the formulation of non-intended computations complicated. However, there are obvious "irregularities": first, there is the principal distinction between type "Cell" and numerical types to which different operations apply. Second, there is a number of special operations (e.g. aggregation operations, cell selection) that could be avoided by making the DSL more general, which contradicts the formal notion of simplicity. Also type completeness is not given since list only have type *Cell* and means of abstraction (e.g. functions) are severely limited by intention, ultimately based on the assumption that CA are inherently simple. In conclusion, characteristics of ECAL that follow from the consideration of explicitness contradict with traditional notions of simplicity and regularity in the context of programming languages.

7 Conclusions and Outlook

7.1 Conclusion

Some conclusions have been drawn according to the different subtopics in this thesis. Now the relation to the highest-level goals stated are considered.

The first goal is the evaluation of the feasibility of the model-driven approach to EMS. This thesis introduced the philosophical-cognitive framework of model-based science as a general conceptual framework for the design of DSLs for EMS that gives relatively specific criteria in contrast to common design criteria of computer languages with respect to pragmatic aspects. At the most general level of evaluation (detailed in conclusions in Chapters 2.5 and 3.6), it has been argued that the foremost characteristic of model-driven approach with relevance to EMS is the efficient provision of coupled DSLs that under the specific forms of uncertainty must be understood as an essential part of distributed cognitive systems. By defining a general form of type hierarchies at the base of reasoning processes as observed in reality, the theory of model-based science shows that type hierarchies which are based on logical formal reasoning (e.g. semantic nets, class hierarchies) only describe that part of scientific reasoning, where reasoning is rather straightforward and uncertainty relatively low, thus they are rather adequate for deductive reasoning and documentation of gatherd knowledge. In this thesis it has been discussed how metamodeled DSLs, when perceived as representations of types, may accommodate reasoning processes under great epistemic uncertainty. The specific characteristic of DSLs is that these combine necessary restrictions with necessary degrees of freedom required for scientific reasoning processes based on short motor-activity cycles and explorative simulation-based investigation of hypothetical mechanisms, which typical formal type hierarchies do not support and programming languages prohibit through implicitness of representation. Although this clearly states metamodeled DSLs as a distinct alternative to formal ontologies for knowledge representation, it gives the metamodels of DSLs, an epistemic status, that would require further consideration and investigation, in particular with respect to its formal representation.

Cellular Automata have been presented as a modeling paradigm that is particularly used for modeling systems with expected chaotic and self-organizing behavior for the purpose of knowledge discovery. Although this poses particular high requirements to transparency of conceptualizations, it has been concluded that there is a heterogeneity of notions that a corresponding class of CA cannot be defined formally, but rather at the base of pragmatic aspects. However, in the implementation presented here, the transfer of concepts and a formalization of type hierarchies in the sense of model-based science is implicit, thus has to be established by modelers cognitively. With ECAL, this thesis proposes a class of CA that particularly aims at supporting the transfer mechanisms from the field of micro-scale physical modeling to the field of macro-scale modeling in EMS, by providing language elements that embody the most characterizing features of micro-scale modeling (e.g. discrete time steps and homogeneity of geometries) with the expressive means required in the context of macro-scale modeling (e.g. groups of cells). The case

study showed the applicability of ECAL and the language-centered approach, based on existing metamodeling and modeling software. Further it has been demonstrated that GIS and simulation can be integrated based on language metamodeling, when GIS and OGC specifications are perceived as a semantic base for language elements of DSLs.

7.2 Outlook

Clearly, the breadth of the topic chosen as the subject of this thesis prohibits comprehensive consideration of all relevant aspects and a sound implementation that reaches beyond prototypical status. In particular, the DSLs except ECAL have been developed as minimal prototypes aligned just with the requirements of the reimplementations and the case study, such that there is rather great potential for a further development of these DSLs. Relevant topics that naturally occur within the presented approach include issues that generally apply to the field of MDE, e.g. the technical composition of language metamodels (including composition of model transformations, code generators, which is related to the issue of combining descriptions of operational semantics), debugging, the identification of reusable patterns and technical support for the evolution of DSLs and coevolution of models. The use of technology independent specifications requires to address issues of non-functional aspects (such as simulator performance) that are a prerequisite for the reuse of models with different framework-level technologies. From a slightly different perspective however, technology independent specifications generally suggest possibilities to the optimization of simulator performance. The evaluation of this aspect would is necessary to conclude comprehensively on the feasibility of LCA. As argued above, object-oriented metamodels suggest a direct representation of general type hierarchies within LCA at least partly, which has not been evaluated yet in practice. Despite these unresolved issues, the relative ease with which DSLs can be defined and brought into action provides a technical framework for the further investigation of issues related to the use of DSLs in EMS.

Appendix A

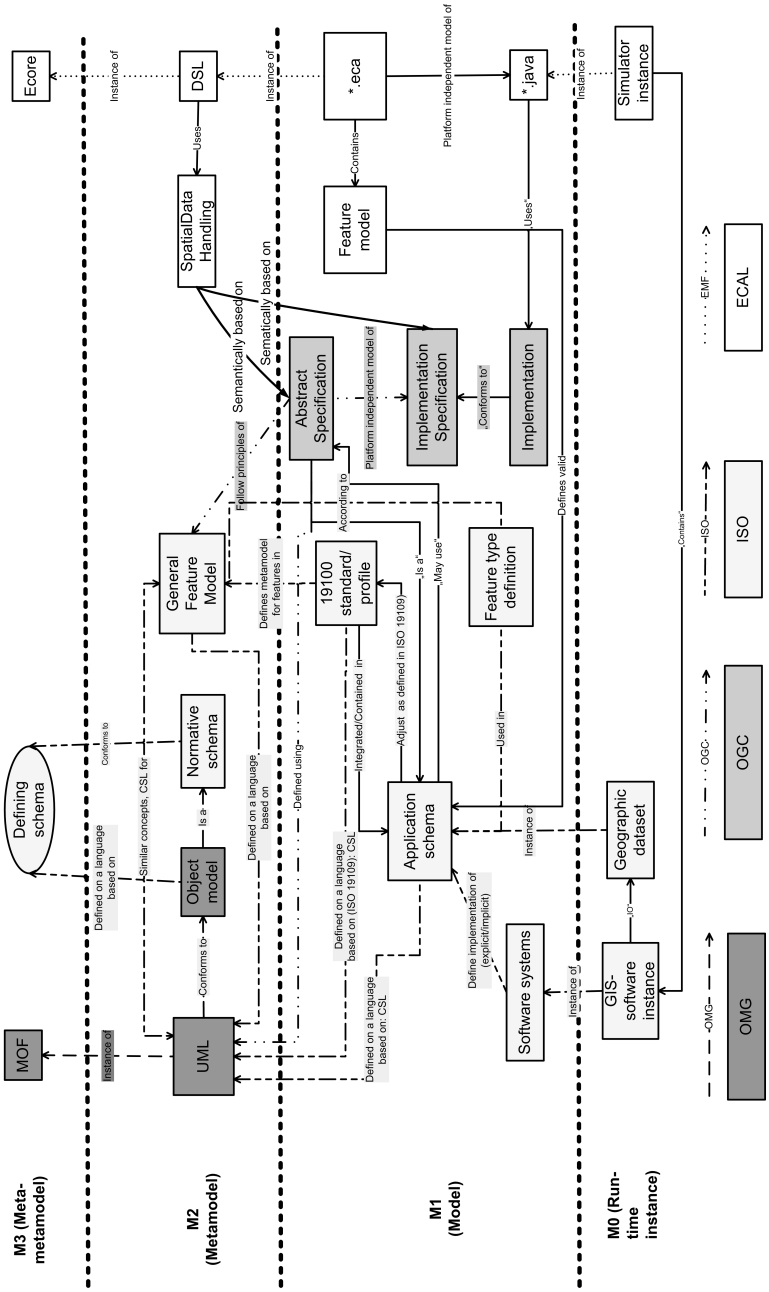

Figure 1: Overview of the relationships of OGC and ISO specifications.

Appendix B

This is the experiment description of one of the experiments conducted with the modified SLEUTH model. The SLEUTH experiments encompass dynamic input into the UGM CA model that updates the data modeling the transportation network and the population density. These are represented by *timeSeries* models that provide the corresponding values at specified times at the output ports. Listing 1 presents their specifications, where each model specifies a number of values for specific times and an output port.

Listing 1: Time series models in the SLEUTH case study in the DSL *timeSeries*.

```
1  CoverageTimeSeriesModel{
2      Name = RoadsModelGen;
3      Time [0.0] Variable GridCoverage <Int> t0 ; Value = SetExternally;
4      Time [12.0] Variable GridCoverage <Int> t1 ; Value = SetExternally;
5      Portname=OP_RoadsTs, Porttype = GridCoverage <Int>;
6  }
7  CoverageTimeSeriesModel{
8      Name = PopDensModelGen;
9      Time [0.0] Variable GridCoverage <Int> Pt0 ; Value = SetExternally;
10     Time [13.0] Variable GridCoverage <Int> Pt1 ; Value = SetExternally;
11     Portname=OP_PopdensTs, Porttype = GridCoverage <Int>;
12 }
```

Listing 2 presents the couplings of models, where the CA model is referenced by its name *SLEUTHEca*.

Listing 2: Model couplings in the SLEUTH case study in the DSL *simCore*.

```
1  ModelCouplings{
2      Name = RoadTimeSeries2Eca, LinkModelOut RoadsModelGen InputModel Eca:
           IP_Roads@SLEUTHEca Synchronisation = PARALLEL
3      Value[ NumericQuery{SELECT [OutputPort(OP_RoadsTs)]} ];
4      Name = PopTimeSeries2Eca, LinkModelOut PopDensModelGen InputModel Eca
           :IP_Pop@SLEUTHEca Synchronisation = PARALLEL
5      Value[ NumericQuery{SELECT [OutputPort(OP_PopdensTs)]} ];
6  }
```

Listing 3 lists the input ports, output ports, input frame and output frame of the UGM modified version in *ECAL*.

Listing 3: Input and output frame and ports of SLEUTH case study in the DSL *ECAL*.

```
1  InputFrame{
2    GlobalLevelCalculation{
3      Transportation=ToCellstate(InputPort(IP_Roads));
4      Popdens=ToCellstate(InputPort(IP_Pop))
5    }
6  }
7
8  OutputFrame{
9    GlobalVariables[
10     Variable Int NumSpontGrowth ;
```

```
11      Variable Int NumEdgeGrowth ;
12      Variable Int NumOrganicGrowth ;
13      Variable Int NumRoadGrowth ;
14    ]
15    GlobalLevelCalculation{
16      OutputPort(OP_Urban)=ToGridCoverage($this->Urban);
17      OutputPort(OP_SpreadCoeff)=SpreadCoeff;
18      OutputPort(OP_BreedCoeff)=BreedCoeff;
19      OutputPort(OP_Growth)=GrowthRateVar;
20      OutputPort(OP_DispersionCoeff)=DispersionCoeff;
21      OutputPort(OP_SlopeCoeff)=SlopeCoeff;
22      OutputPort(OP_RoadCoeff)=RoadCoeff;
23      OutputPort(OP_SpontGrowth)=NumSpontGrowth;
24      OutputPort(OP_EdgeGrowth)=NumEdgeGrowth;
25      OutputPort(OP_OrganicGrowth)=NumOrganicGrowth;
26      OutputPort(OP_RoadGrowth)=NumRoadGrowth
27    }
28  }
29
30  InputPorts{
31    Name = IP_Roads, DataType = GridCoverage <Int>;
32    Name = IP_Pop, DataType = GridCoverage <Int>;
33  }
34
35  OutputPorts{
36    Output { Name = OP_Slope, DataType = GridCoverage <Real>};
37    Output { Name = OP_Urban, DataType = GridCoverage <Int>};
38    Output { Name = OP_Transportation, DataType = GridCoverage <Int>};
39    Output { Name = OP_Landuse, DataType = GridCoverage <Int>};
40    Output { Name = OP_Excluded, DataType = GridCoverage <Int>};
41    Output { Name = OP_SpreadCoeff, DataType = Real};
42    Output { Name = OP_BreedCoeff, DataType = Real};
43    Output { Name = OP_Growth, DataType = Real};
44    Output { Name = OP_DispersionCoeff, DataType = Real};
45    Output { Name = OP_SlopeCoeff, DataType = Real};
46    Output { Name = OP_RoadCoeff, DataType = Real};
47    Output { Name = OP_SpontGrowth, DataType = Int};
48    Output { Name = OP_EdgeGrowth, DataType = Int};
49    Output { Name = OP_OrganicGrowth, DataType = Int};
50    Output { Name = OP_RoadGrowth, DataType = Int};
51  }
```

Listing 4: Experiment description of the SLEUTH model in *simDescription*.

```
1  Experiment:
2    InputValues {
3    ConstantInput [ Coverage "SLEUTHEca"::"Slope" ; InitValue =
         GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/slope_gt.tif
         "];
4    ConstantInput [ Coverage "SLEUTHEca"::"Excluded" ; InitValue =
         GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/exclusion_gt.
         tif"];
5    ConstantInput [ Coverage "SLEUTHEca"::"Urban" ; InitValue =
         GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/urb_88_ndvi_gt
         .tif"];
6    ConstantInput [ Coverage "SLEUTHEca"::"Transportation" ; InitValue =
         GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/roads_88_gt.
         tif"];
```

```
7  ConstantInput [ Coverage "SLEUTHEca"::"Popdens" ; InitValue =
       GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/pop89_gt.tif
       "];
8  ConstantInput [ Real "SLEUTHEca"::"CRITICAL_HIGH" ; InitValue = 1.3];
9  ConstantInput [ Real "SLEUTHEca"::"CRITICAL_LOW" ; InitValue = 0.97];
10 ConstantInput [ Real "SLEUTHEca"::"Seed" ; InitValue = 1.0];
11 ConstantInput [ Real "SLEUTHEca"::"BOOM" ; InitValue = 1.01];
12 ConstantInput [ Real "SLEUTHEca"::"BUST" ; InitValue = 0.09];
13 ConstantInput [ Real "SLEUTHEca"::"SlopeSensitivity" ; InitValue =
       0.1];
14 ConstantInput [ Real "SLEUTHEca"::"RoadSensitivity" ; InitValue =
       0.01];
15 ConstantInput [ Real "SLEUTHEca"::"MIN_SLOPE_RESISTANCE" ; InitValue =
       0.01];
16 ConstantInput [ Real "SLEUTHEca"::"MIN_ROAD_GRAVITY" ; InitValue =
       0.01];
17 ConstantInput [ Real "SLEUTHEca"::"MIN_DIFFUSION" ; InitValue = 0.01];
18 ConstantInput [ Real "SLEUTHEca"::"MIN_SPREAD" ; InitValue = 0.01];
19 ConstantInput [ Real "SLEUTHEca"::"MIN_BREED" ; InitValue = 0.01];
20 ConstantInput [ Real "SLEUTHEca"::"MAX_SLOPE_RESISTANCE" ; InitValue =
       100.0];
21 ConstantInput [ Real "SLEUTHEca"::"MAX_ROAD_GRAVITY" ; InitValue =
       100.0];
22 ConstantInput [ Real "SLEUTHEca"::"MAX_DIFFUSION" ; InitValue = 100.0];
23 ConstantInput [ Real "SLEUTHEca"::"MAX_SPREAD" ; InitValue = 100.0];
24 ConstantInput [ Real "SLEUTHEca"::"MAX_BREED" ; InitValue = 100.0];
25 ConstantInput [ Real "SLEUTHEca"::"CRITICAL_MAX_POP" ; InitValue =
       500.0];
26 ConstantInput [ Real "SLEUTHEca"::"CRITICAL_SLOPE" ; InitValue = 15.0];
27 ConstantInput [ Real "SLEUTHEca"::"MAX_ROAD_VALUE" ; InitValue = 10.0];
28 ConstantInput [ Coverage "RoadsModelGen"::"t0" ; InitValue =
       GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/roads_88_gt.
       tif"];
29 ConstantInput [ Coverage "RoadsModelGen"::"t1" ; InitValue =
       GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/roads_00_gt.
       tif"];
30 ConstantInput [ Coverage "PopDensModelGen"::"Pt0" ; InitValue =
       GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/pop89_gt.tif
       "];
31 ConstantInput [ Coverage "PopDensModelGen"::"Pt1" ; InitValue =
       GridCoverageFromFile "./cainputdata/SLEUTH/SLEUTH_GT/pop01_gt.tif
       "];
32 ConstantInput [ Real "SLEUTHEca"::"DispersionCoeff" ; InitValue = 1.0];
33 ConstantInput [ Real "SLEUTHEca"::"BreedCoeff" ; InitValue = 94.0];
34 ConstantInput [ Real "SLEUTHEca"::"SpreadCoeff" ; InitValue = 83.0];
35 ConstantInput [ Real "SLEUTHEca"::"SlopeCoeff" ; InitValue = 7.0];
36 ConstantInput [ Real "SLEUTHEca"::"RoadCoeff" ; InitValue = 78.0];
37 ConstantInput [ Real "SLEUTHEca"::"PopCoeff" ; InitValue = 55.0];
38 VaryInitialStateStepwise : "SLEUTHEca"::"CRITICAL_MIN_POP" , min : Real
       (0.0), max : Real(500.0), step : Real(50.0);}
39 MonteCarlo {name = mc, runs = 5}
40 Observation : name =obs_urban, model ="SLEUTHEca", port ="OP_Urban",
       datatype = GridCoverage <Int>;
41 Observation : name =breedcoeff, model ="SLEUTHEca", port ="
       OP_BreedCoeff", datatype = Real;
42 Observation : name =dispersioncoeff, model ="SLEUTHEca", port ="
       OP_DispersionCoeff", datatype = Real;
43 Observation : name =spreadcoeff, model ="SLEUTHEca", port ="
       OP_SpreadCoeff", datatype = Real;
```

```
44  Observation : name =slopecoeff, model ="SLEUTHEca", port ="
        OP_SlopeCoeff", datatype = Real;
45  Observation : name =roadcoeff, model ="SLEUTHEca", port ="OP_RoadCoeff
        ", datatype = Real;
46  Observation : name =growthrate, model ="SLEUTHEca", port ="OP_Growth",
        datatype = Real;
```

Listing 5: Analysis of observed data in SLEUTH with *GISDSL*

```
1   Analysis {
2     CompareGridCoverage [LEESALEE] {
3       Output : name = LeeSaleeSingleRun , Datatype = Real;
4       Input : obs_urban, Datatype = GridCoverage <Int>;
5       Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
6       Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
7     };
8
9     CompareGridCoverage [OVERALLACCURACY] {
10     Output : name = OASingleRun , Datatype = Real;
11     Input : obs_urban, Datatype = GridCoverage <Int>;
12     Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
           urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
13     Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
           urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
14    };
15
16    CompareGridCoverage [PRODUCERACCURACY] {
17      Output : name = PASingleRun , Datatype = Real;
18      Input : obs_urban, Datatype = GridCoverage <Int>;
19      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
20      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
21    };
22
23    CompareGridCoverage [PRODUCERACCURACYNEG] {
24      Output : name = PANotSingleRun , Datatype = Real;
25      Input : obs_urban, Datatype = GridCoverage <Int>;
26      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
27      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
28    };
29
30    CompareGridCoverage [USERACCURACY] {
31      Output : name = UASingleRun , Datatype = Real;
32      Input : obs_urban, Datatype = GridCoverage <Int>;
33      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
34      Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
            urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
35    };
36
37    CompareGridCoverage [USERACCURACYNEG] {
38      Output : name = UANotSingleRun , Datatype = Real;
39      Input : obs_urban, Datatype = GridCoverage <Int>;
```

```
40        Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
              urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
41        Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
              urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
42      };
43
44      CompareGridCoverage [KAPPA] {
45        Output : name = KappaSingleRun , Datatype = Real;
46        Input : obs_urban, Datatype = GridCoverage <Int>;
47        Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
              urb_00_ndvi_gt.tif"] With [Time(12.0)] ;
48        Analysis : Compare [CoverageFile :"./cainputdata/SLEUTH/SLEUTH_GT/
              urb_07_ndvi_gt.tif"] With [Time(19.0)] ;
49      };
50
51      AggregateByMean {
52        Output : name = LeeSaleeMean , Datatype = Real;
53        Input : LeeSaleeSingleRun, Datatype = Real, AggregateOver : mc;
54      };
55
56      AggregateByMean {
57        Output : name = ProducerAccuracyMean , Datatype = Real;
58        Input : PASingleRun, Datatype = Real, AggregateOver : mc;
59      };
60
61      AggregateByMean {
62        Output : name = ProducerAccuracyNotMean , Datatype = Real;
63        Input : PANotSingleRun, Datatype = Real, AggregateOver : mc;
64      };
65
66      AggregateByMean {
67        Output : name = UserAccuracyMean , Datatype = Real;
68        Input : UASingleRun, Datatype = Real, AggregateOver : mc;
69      };
70
71      AggregateByMean {
72        Output : name = UserAccuracyNotMean , Datatype = Real;
73        Input : UANotSingleRun, Datatype = Real, AggregateOver : mc;
74      };
75
76      AggregateByMean {
77        Output : name = KappaMean , Datatype = Real;
78        Input : KappaSingleRun, Datatype = Real, AggregateOver : mc;
79      };
80
81      AggregateByMean {
82        Output : name = OAMean , Datatype = Real;
83        Input : OASingleRun, Datatype = Real, AggregateOver : mc;
84      };
85  }
```

The observed data (obs_urban) is aggregated for calibration by means of an analysis that calculates a number of accuracy measures. In addition to the LeeSalee index in the original version, we also calculate: Overall Accuracy, Kappa Coefficient, and Producer Accuracy and User Accuracy for each of the two land use classes (urbanized, not urbanized). In contrast to most of the accuracy measures used in the original implementation, these measures can be calculated by comparisons of two single data sets. The listing shows the specification of the analysis using the analysis DSL. It takes the observation (obs_urban)

as input. Two analysis steps are specified. The first step specifies the computation of the Lee-Salee index by means of a *CompareGridCoverage* operator with *obs_urban* as input and *LeeSaleeSingleRun* as output. The analysis is executed for two times only (12.0, 19.0), where for each time a particular data set that represents measured data is used for comparison. The operator is parameterized with LEESALEE. The next step of the analysis is the aggregation of the result by averaging over all multi-runs (*mc*) for each time step separately. The result of the aggregation is identified by its name *LeeSaleeMean* and is used in further analysis steps.

Appendix C

This appendix presents the reimplementation of the CA model for modeling fire spread and fire fighting in urban areas published in (Ohgai et al., 2007). It combines a fire spread process with the process of firefighting. It is a cyclic CA where the state variable *Burnstate* models the phases: 0=Unburnable, 1=Not burning, 2=Catching fire, 3=Burning and 4=Extinguished. Fire spread is modeled probabilistically, where the neighborhood and the probability of spread is dependent on wind speed and direction and spread modeled by means of a nested *ForEachCell* directives (ll. 78 - 119). The cyclic behavior is described in lines 150 - 160.

The firefighting is modeled such that for each burning cell, cells with water supply in the neighborhood are considered as water sources (ll. 125 - 128). Then all water sources select a number of burning cells in their neighborhood as targets for water depending on the water content (ll. 130 - 139). Then reciepient cells start to receive water (ll. 141 - 148).

Listing 6: Reimplementation of fire spread model with *ECAL*

```
1 Eca (name: Ohgai2007Eca width: 254 height: 173 Boundary:CUT stepsize: 1.0
      )
2     SpatialReference : RasterFile Directory = "./cainputdata/" Filename =
          "ohgai_s_small2" FileExt = "tif"  ;
3     GlobalStateVariables
4     {
5         State Parameter Real Alpha ;
6         State Parameter Real Beta ;
7         State Parameter Real Windspeed ;
8         State Parameter Real Winddirection ;
9         State Parameter Real t1 ;
10        State Parameter Real t2 ;
11        State Parameter Real Cw ;
12        State Parameter Int tf ;
13        State Parameter Real Rw ;
14        State Variable List <Cell> BurningCells ;
15        State Variable List <Cell> WaterSources ;
16    }
17
18    CellDefinition
19    {
20        Cellstate Variable Int Burnstate ;
21        Cellstate Parameter Real S ;
22        Cellstate Parameter Real P ;
23        Cellstate Variable Real t0 ;
24        Cellstate Variable Real StartRecieveWater ;
25        Cellstate Parameter Real WaterCap ;
26        Cellstate Variable List <Cell> DirectNeighbors ;
27        Cellstate Variable List <Cell> wsrcNB ;
28    }
29    Initialisation{
30        ForEachCell In [AllCells] INSTANT{
```

```
31              $this->DirectNeighbors=CellsWithinRadius ( 3.5 ) EXCLUSIVE ;
32              $this->wsrcNB=CellsWithinRadius ( 9.5 ) EXCLUSIVE ;
33              $this->t0=0 - 1;
34              $this->StartRecieveWater=0 - 1
35          }
36          GlobalTransition{
37              Alpha=SetExternally;
38              Beta=SetExternally;
39              Windspeed=SetExternally;
40              Winddirection=SetExternally;
41              t1=SetExternally;
42              t2=SetExternally;
43              Cw=SetExternally;
44              tf=SetExternally;
45              Rw=SetExternally;
46              Burnstate=SetExternally;
47              S=SetExternally;
48              WaterCap=SetExternally;
49              P=SetExternally
50          }
51      }
52      EcaTransitions{
53          GlobalTransition{
54              BurningCells=SelectCells From {AllCells}  Having [$this->
                    Burnstate == 3]
55          }
56
57          ForEachCell In [BurningCells]  INSTANT{
58              Variable Real tckl = 0;
59              Variable Real burntime = 0;
60              Variable Real factor = 0;
61              tckl=Simtime - $this->t0;
62              burntime=t2 - t1;
63              factor=burntime / 5;
64              if[tckl >= t1 AND tckl <= factor + t1]
65              then[
66              $this->P=4.0 / (t2 - t1) * tckl + (0.2 + t2 - 4.2 * t1) / (t2
                    - t1)];
67              if[tckl >= factor + t1 AND tckl <= t2]
68              then[
69              $this->P=5.0 / (4.0 * (t2 - t1)) * (t2 - tckl)]
70          }
71
72          GlobalTransition{
73              BurningCells=SelectCells From {AllCells}  Having [$this->
                    Burnstate == 3]
74          }
75
76          ForEachCell In [BurningCells] AS src NONE{
77
78                  ForEachCell In [$src->DirectNeighbors] AS rec NONE{
79                      Variable Real RelativeWindDirection = 0;
80                      Variable Real WindAngle = 0;
81                      Variable Real RelativeWindPos = 0;
82                      Variable Real W = 0;
83                      Variable Real FSJI = 0;
84                      RelativeWindDirection=Abs(Direction(rec, src) -
                            Winddirection );
85                      RelativeWindPos=Min (RelativeWindDirection, 360 -
```

```
                    RelativeWindDirection);
86                  if[Windspeed >= 6 AND Windspeed <= 8 AND
                        RelativeWindPos > 90.1 AND Distance(rec, src) >
                        2.83]
87                  then[
88                  W=0.05];
89                  if[Windspeed >= 6 AND Windspeed <= 8 AND
                        RelativeWindPos > 90.1 AND Distance(rec, src) >
                        1.9 AND Distance(rec, src) < 2.83]
90                  then[
91                  W=0.1];
92                  if[Windspeed >= 6 AND Windspeed <= 8 AND
                        RelativeWindPos > 90.1 AND Distance(rec, src) <
                        1.5]
93                  then[
94                  W=0.2];
95                  if[Windspeed >= 6 AND Windspeed <= 8 AND Distance(rec
                        , src) < 1.1 AND RelativeWindPos < 90.1 AND
                        RelativeWindPos > 89.9]
96                  then[
97                  W=0.5];
98                  if[Windspeed >= 6 AND Windspeed <= 8 AND Distance(rec
                        , src) < 2.1 AND Distance(rec, src) > 1.1 AND
                        RelativeWindPos < 90.1 AND RelativeWindPos >
                        89.9]
99                  then[
100                 W=0.4];
101                 if[Windspeed >= 6 AND Windspeed <= 8 AND Distance(rec
                        , src) < 3.1 AND Distance(rec, src) > 2.1 AND
                        RelativeWindPos < 90.1 AND RelativeWindPos >
                        89.9]
102                 then[
103                 W=0.3];
104                 if[Windspeed >= 6 AND Windspeed <= 8 AND Distance(rec
                        , src) > 2.81 AND RelativeWindPos < 90.1]
105                 then[
106                 W=0.3];
107                 if[Windspeed >= 6 AND Windspeed <= 8 AND
                        RelativeWindPos < 89.9 AND Distance(rec, src) >
                        1.9 AND Distance(rec, src) < 2.83]
108                 then[
109                 W=0.6];
110                 if[Windspeed >= 6 AND Windspeed <= 8 AND
                        RelativeWindPos < 89.9 AND Distance(rec, src) <
                        1.5]
111                 then[
112                 W=1.0];
113                 FSJI=Alpha * ($rec->S * $rec->P) * W * $src->P;
114                 if[FSJI > RandomReal(0, 1) AND $rec->Burnstate == 1]
115                 then[
116                 $rec->Burnstate=2;
117                 $rec->t0=Simtime]
118             }
119         }
120
121         GlobalLevelCalculation{
122         BurningCells=SelectCells From {AllCells}  Having [$this->
                Burnstate == 2 OR $this->Burnstate == 3]
123         }
```

```
124
125         ForEachCell In [BurningCells] NONE{
126             Variable List <Cell> cl = CellsWithinRadius ( 9.5 ) EXCLUSIVE
                  ;
127             WaterSources=Union [WaterSources, SelectCells From {cl}
                    Having [$this->WaterCap >= 40]]
128         }
129
130         ForEachCell In [WaterSources] AS srcCell INSTANT{
131             Variable Real Ce = 0;
132             Variable List <Cell> recipients ;
133             Ce=Cut($srcCell->WaterCap / (Cw * tf) * Rw,0);
134             recipients=CellsWithinRadius ( 9.5 ) EXCLUSIVE ;
135             recipients=SelectCells From {recipients} Having [$this->
                    Burnstate == 2 OR $this->Burnstate == 3];
136             recipients=Sort (recipients AS recCell By [Distance(srcCell,
                    recCell)] );
137             recipients=SelectCells From {recipients} Having [
                    ListPosition of (this) in (recipients) <= Ce];
138             $this->wsrcNB=recipients
139         }
140
141         ForEachCell In [WaterSources] AS srcC INSTANT{
142
143                 ForEachCell In [$srcC->wsrcNB] AS recC INSTANT{
144                     if[$recC->StartRecieveWater < 0]
145                     then[
146                     $recC->StartRecieveWater=Simtime]
147                 }
148         }
149
150         ForEachCell In [AllCells] NONE{
151             if[$this->Burnstate == 2 AND Simtime - $this->t0 >= t1]
152             then[
153             $this->Burnstate=3];
154             if[$this->Burnstate == 3 AND Simtime - $this->t0 >= t2 - t1]
155             then[
156             $this->Burnstate=4];
157             if[($this->Burnstate == 2 OR $this->Burnstate == 3) AND $this
                    ->StartRecieveWater > 0 AND Simtime - $this->
                    StartRecieveWater >= tf]
158             then[
159             $this->Burnstate=5]
160         }
161     }
162     OutputFrame{
163
164         GlobalLevelCalculation{
165         OutputPort(burn1)=Count( Cells In [AllCells] Having [$this->
                Burnstate == 1] );
166         OutputPort(burn2)=Count( Cells In [AllCells] Having [$this->
                Burnstate == 2] );
167         OutputPort(burn3)=Count( Cells In [AllCells] Having [$this->
                Burnstate == 3] );
168         OutputPort(burn4)=Count( Cells In [AllCells] Having [$this->
                Burnstate == 4] );
169         OutputPort(burn5)=Count( Cells In [AllCells] Having [$this->
                Burnstate == 5] );
170         OutputPort(burnOutCoverage)=ToGridCoverage($this->Burnstate);
```

```
171             OutputPort(burnOut)=ToCellData($this->Burnstate);
172             OutputPort(SOut)=ToCellData($this->S);
173             OutputPort(WatercapOut)=ToCellData($this->WaterCap)
174             }
175         }
176         InputPorts{     }
177         OutputPorts{
178         Output { Name = burnOutCoverage, DataType = GridCoverage <Int>};
179         Output { Name = burnOut, DataType = List <CellData:Real>};
180         Output { Name = SOut, DataType = List <CellData:Real>};
181         Output { Name = WatercapOut, DataType = List <CellData:Real>};
182         Output { Name = burn1, DataType = Int};
183         Output { Name = burn2, DataType = Int};
184         Output { Name = burn3, DataType = Int};
185         Output { Name = burn4, DataType = Int};
186         Output { Name = burn5, DataType = Int};
187         }
188 EndEca
```

Appendix D

This is the reimplementation of the CA fire spread model as published in Alexandridis et al. (2008). The model is basically cyclic in that the state variable (*Burnstate*) values denotes a succession of states (2="fueled", 3="burning",4="burned down"), where succession is described by rules 0-3 (ll. 63 - 73). Further, it contains an explicit spread of fire "Rule 4" (ll. 76 - 102) modeled by nested *ForEachCell* directives. "Rule 5" (ll. 104 - 144) describes a spotting process, thus the transport of burning material by wind. The spotting is described as a process where from a burning cell *sprSrc* (l. 105), possible target cells are selected depending on wind direction and speed (ll. 118 - 126) and for each target it is decided probabilistically, if it catches fire (ll. 129 - 143).

Listing 7: Reimplementation of fire spread model with *ECAL*

```
1  Eca (name: Alexandridis2008Eca width: 385 height: 369  Boundary:CUT
       stepsize: 1.0 )
2
3      SpatialReference : RasterFile Directory = "./cainputdata/Alexandridis
           /" Filename = "elevation2" FileExt = "tif"  ;
4      GlobalStateVariables
5      {
6          State Parameter Real Ph ;
7          State Parameter Real Winddirection ;
8          State Parameter Real Windspeed ;
9          State Parameter Real Pden1 ;
10         State Parameter Real Pden2 ;
11         State Parameter Real Pden3 ;
12         State Parameter Real Pveg1 ;
13         State Parameter Real Pveg2 ;
14         State Parameter Real Pveg3 ;
15         State Parameter Real C1 ;
16         State Parameter Real C2 ;
17         State Parameter Real A ;
18         State Parameter Real Pc0 ;
19         State Parameter Int StartX ;
20         State Parameter Int StartY ;
21     }
22
23     CellDefinition
24     {
25         Cellstate Parameter Real Elevation ;
26         Cellstate Parameter Int VegType ;
27         Cellstate Parameter Int VegDens ;
28         Cellstate Variable Int Burnstate ;
29         Cellstate Variable List <Cell> Neighbors ;
30     }
31     Initialisation{
32         ForEachCell In [AllCells] INSTANT_SERIAL{
33             $this->Neighbors=Name = MOORE EXCLUSIVE
34         }
35         GlobalTransition{
```

```
36              Elevation=SetExternally;
37              VegType=SetExternally;
38              VegDens=SetExternally;
39              Burnstate=SetExternally;
40              Pden1=SetExternally;
41              Pden2=SetExternally;
42              Pden3=SetExternally;
43              Pveg1=SetExternally;
44              Pveg2=SetExternally;
45              Pveg3=SetExternally;
46              C1=SetExternally;
47              C2=SetExternally;
48              A=SetExternally;
49              Pc0=SetExternally;
50              StartX=SetExternally;
51              StartY=SetExternally;
52              Ph=SetExternally;
53              Winddirection=SetExternally;
54              Windspeed=SetExternally
55          }
56      }
57      EcaTransitions{
58        <"Fire spread transition function">
59        GlobalTransition{
60        Variable List <Cell> BurningCells ;
61        if[Simtime == 1]then[BurningCells=SelectCells From {AllCells}
              Having [X(this) == StartX AND Y(this) == StartY]];
62        if[Simtime > 1]then[BurningCells=SelectCells From {AllCells}
              Having [$this->Burnstate == 3]];
63        <"Rule0">
64        ForEachCell In [BurningCells] INSTANT{
65          $this->Burnstate=3
66        };
67        <"Rule 1,2,3">
68        if[Simtime > 1]
69        then[
70          ForEachCell In [BurningCells] NONE{
71            $this->Burnstate=4
72          }
73        ];
74
75        <"Rule 4">
76        ForEachCell In [BurningCells] AS ssrc NONE{
77          ForEachCell In [$ssrc->Neighbors] AS srec NONE{
78            Variable Real Pburn ;
79            Variable Real Pveg ;
80            Variable Real Pdens ;
81            Variable Real Pw ;
82            Variable Real Ps ;
83            Variable Real Ft ;
84            Variable Real SpreadAngle ;
85            Variable Real WindSpreadAngle ;
86            Variable Real SlopeAngle ;
87            if[$srec->VegType == 1]then[Pveg=0 - Pveg1];
88            if[$srec->VegType == 2]then[Pveg=Pveg2];
89            if[$srec->VegType == 3]then[Pveg=Pveg3];
90            if[$srec->VegDens == 1]then[Pdens=0 - Pden1];
91            if[$srec->VegDens == 2]then[Pdens=Pden2];
92            if[$srec->VegDens == 3]then[Pdens=Pden3];
```

```
93            SpreadAngle=Direction(ssrc, srec);
94            WindSpreadAngle=Winddirection - SpreadAngle;
95            Ft=Exp(Windspeed * C2 * (Cos(WindSpreadAngle) - 1));
96            Pw=Exp(Windspeed * C1) * Ft;
97            SlopeAngle=1 / Tan(($ssrc->Elevation - $srec->Elevation) /
                  Distance(ssrc, srec));
98            Ps=Exp(SlopeAngle);
99            Pburn=Ph * (1 + Pveg) * (1 + Pdens) * Pw * Ps;
100           if[$srec->Burnstate == 2 AND RandomReal(0, 1) <= Pburn]then[ $
                  srec->Burnstate=3]
101         }
102       };
103
104       <"Rule 5">
105       ForEachCell In [BurningCells] AS spSrc INSTANT{
106         Variable Real NumCones ;
107         Variable List <Cell> SpottingNbrs ;
108         SpottingNbrs=NewList (Cell);
109         NumCones=RandomPoisson(3.0);
110
111         GlobalTransition{
112           Variable Int SpotDirection ;
113           Variable Real SpotDistance ;
114           Variable Real Thrust ;
115           Variable Real WindAngle ;
116           Variable Real OffsetX ;
117           Variable Real OffsetY ;
118           SpotDirection=RandomIntUniform(0, 360);
119           WindAngle=SpotDirection - Winddirection;
120           Thrust=RandomNormal(7, 5);
121           SpotDistance=Thrust * Exp(Windspeed * (Cos(WindAngle) - 1));
122           OffsetX=Cut(Sin(SpotDirection) * SpotDistance,0);
123           OffsetY=Cut(Cos(SpotDirection) * SpotDistance,0);
124           SpottingNbrs=AddElement [SelectCell Having [dX=OffsetX, dY=(0 -
                  1) * OffsetY]] To [SpottingNbrs];
125           NumCones=NumCones - 1
126         RepeatUntil{NumCones < 0}
127         };
128
129         ForEachCell In [SpottingNbrs] AS spRec INSTANT{
130           Variable Real Pc ;
131           Variable Real Pcd ;
132           Variable Real Pctype ;
133           Variable Real Pcdens ;
134           if[$spRec->VegType == 1]then[Pctype=0 - Pveg1];
135           if[$spRec->VegType == 2]then[Pctype=Pveg2];
136           if[$spRec->VegType == 3]then[Pctype=Pveg3];
137           if[$spRec->VegDens == 1]then[Pcdens=0 - Pden1];
138           if[$spRec->VegDens == 2]then[Pcdens=Pden2];
139           if[$spRec->VegDens == 3]then[Pcdens=Pden3];
140           Pcd=Pcdens * Pctype;
141           Pc=Pc0 * (1 + Pcd);
142           if[$spRec->Burnstate == 2 AND RandomReal(0, 1) <= Pc]then[$
                  spRec->Burnstate=3]
143         }
144       }
145     }
146   }
147
```

```
148   InputFrame{ }
149   OutputFrame{
150     GlobalLevelCalculation{
151       OutputPort(OP_Burnstate)=ToGridCoverage($this->Burnstate);
152       OutputPort(OP_Vegtype)=ToGridCoverage($this->VegType);
153       OutputPort(OP_Vegdens)=ToGridCoverage($this->VegDens);
154       OutputPort(OP_Elevation)=ToGridCoverage($this->Elevation)
155     }
156   }
157
158  InputPorts{     }
159
160  OutputPorts{
161   Output { Name = OP_Burnstate, DataType = GridCoverage <Int>};
162   Output { Name = OP_Vegtype, DataType = GridCoverage <Int>};
163   Output { Name = OP_Vegdens, DataType = GridCoverage <Int>};
164   Output { Name = OP_Elevation, DataType = GridCoverage <Real>};
165  }
166 EndEca
```

Abbreviations & Acronyms

API Application Programming Interface, page 74

ASM Abstract State Machine, page 67

AST Abstract Syntax Tree, page 64

BNF Backus-Naur Form, page 66

CAMPaM Computer Automated Multi-paradigm Modeling, page 79

CPU Central Processing Unit, page 65

DAE Differential-algebraic Equation, page 35

DEVS Discrete Event System Specification, page 41

DSL Domain-specific Language, page 69

EMS Environmental Modeling and Simulation, page 5

FSSP Firing-squad synchronization problem, page 112

FTG Formalism Transformation Graph, page 81

GAS Geographic Automata System, page 39

GIS Geographic Information System, page 39

GOL Game of Life, page 114

GPL General-purpose Programming Language, page 69

GST General Systems Theory, page 7

HLA High Level Architecture, page 79

IDE Integrated Development Environment, page 64

IMA Integrated Modeling Architecture, page 87

IMT Integrated Modeling Tool, page 87

ISO International Standardization Organization, page 44

LBM Lattice Boltzmann Method, page 124

LGA Lattice Gas Automaton, page 124

M2M Model-to-model transformation, page 91

M2T Model-to-text transformation, page 91

M&S Modeling and Simulation, page 5

MDA Model Driven Architecture, page 91

MDE Model-driven Engineering, page 64

MML Metamodeling language, page 98

NCCA Number-conserving Cellular Automata, page 114

NMR Nuclear Magnetic Resonance, page 112

OCA One-way Cellular Automata, page 112

ODE Ordinary Differential Equations, page 33

OGC Open Geospatial Consortium, page 44

OMG Object Management Group, page 91

PDE Partial Differential Equations, page 33

QCA Quantum Cellular Automata, page 112

RUP Rational Unified Process, page 91

SOS Structural Operational Semantics, page 67

UML Unified Modeling Language, page 22

Glossary

Abstract syntax The abstract syntax of a computer language refers to representations of utterances of a computer language without purely syntactical elements., 65

Application Programming Interface Specification of how a software system can be accessed by other programs at the level of source code, 74

Attractor Invariant subset of the phase space of a Dynamical System towards the state of a Dynamical System converges from within a corresponding neighborhood (basin of attraction). Typical attractors are limit points, limit cycles and strange attractors., 17

Autocatalytic reaction Chemical reaction where one of the reactants is also a product of the reaction. The rate equation is typically non-linear, 124

Cell space Cellular Automaton, 111

Cellular Automaton A Dynamical System specified as a lattice of homogeneous cells, with each cell having a state and a neighborhood and the dynamics depending on local interactions between neighboring cells, 111

Class In object-orientation, a class is an abstraction that can be used to instantiate objects, 19

Class hierarchy An arrangement of abstractions that models an object-oriented system, 20

Class of models A set of models where all models of a class share relevant properties. A class of models may specified by a single specification., 48

Closed system A system without relevant interaction with the system environment., 8

Co-simulation Simulation where different simulators are executed in parallel and may exchange data, 77

Combined simulation Combined discrete and continuous simulation (time and state), 45

Combined systems Dynamical system that combines continuous and discrete-time behavior., 27

Complex system System to which complexity is attributed referring to a set of perceptions (e.g. hierarchical composition, non-linearity, missing observations etc.) that cause difficulties in establishing characterizing features, 24

Component Software component, 84

Component platform Software that provides services (e.g. management, execution, synchronization) that can be used to execute a number of *software components*, 77

Component-based simulation Simulation where simulator is composed of *model components*, 77

Computational workflow A computation that is defined as a set of ordered computational tasks that are connected via data exchange., 77

Computer language Means to express informational content to be processed by a computer. Computer languages are characterized by means of their *syntax* and *semantics*., 65

Concrete syntax The concrete syntax represents the utterances of a computer language as they actually appear to sender and receiver in terms of related symbols., 65

Continuous Dynamical System Dynamical System that is perceived as changing its state discontinuously a finite number of times in any given time span, with a possibly finite number of possible discrete states, 27

Correctness (of a simulator) Property of a simulator that it simulates the behavior of a *model* correctly by correct generation of state trajectory from a given initial state and inputs. Correctness is implemented and validated by *verification*, 48

Data abstraction Conceptual framework that encourages the decomposition of specification of software by means of abstractions that define data types as structured sets of values and associated operations (e.g. object-oriented classes), 70

Debugging Process of identifying and elimination of errors in a program, 90

Declarative modeling language DSL, 82

Deterministic system Dynamical system of which the behavior is completely predetermined by the initial state, 27

Digital systems analysis *Systems analysis* using digital computers to simulate a system's behavior, 9

Discrete-event systems Dynamical system that is perceived as changing its state at a finite number of arbitrary times as a consequence of other events, states or time passed, 27

Discrete-time system Dynamical System that is perceived as changing its state discontinuously a finite number of times in any given time span, with a possibly finite number of possible discrete states, 27

Domain-specific language (DSL) Computer language that is designed to be universal within a specific-problem domain by means of provision of domain-specific abstractions that might sacrifice expressive power for the sake of expressivity, 69

Dynamic semantics The dynamic semantics of a language is the meaning of utterances of a language in terms of computations. It is theoretically geiven by means of a mapping from syntax to a semantic domain., 65

Dynamical System Mathematical model focusing on the evolution of processes with time. A dynamical system consist of a time base, a phase space and an evolution function, 12

Ecore Implementation of the object-oriented meta-metamodel EMOF within the Eclipse Modeling Framework, 94

Environment The subject of environmental science defined as the physical non-living and living environment of organisms., 5

Environmental Modeling and Simulation Application of M&S to the characterization of environmental systems within environmental science, 6

Equilibrium State of a Dynamical System that does not change under application of δ typically referred to as fixed points for maps or stationary point for continuous Dynamical Systems, 17

Experiment Controlled generation and collection of observations of a system or a model of a system and their successive evaluation for the purpose of characterization of the system or model, 46

Experiment series A set of related experiments, 49

Experimental frame Formal specification of the conditions under which the system/model is observed or experimented with, 49

Experimentation The controlled variation of a system's environment and observation of the respective systems' state., 9

Expressive power The expressive power of a computer language refers to the capability of a language to use the features of the receiving computer in terms of possible computations, 69

Expressivity The expressivity of a computer language is the degree to which a computer language provides problem-specific abstractions that facilitate a compact representation of computations, 69

External DSL DSL where the language concepts are provided by means of its own syntax and semantics., 72

Federated simulation Simulation where several simulators are coupled at runtime via data exchange, 77

FHP-model The Frisch, Hasslacher and Pomeau lattice gas model which is a variation of the HPP model with hexagonal lattice, 125

Fixed point $q^* \in Q$ such that $\delta(q^*) = q^*$. Fixed points might be stable or unstable, 17

Formal grammar Compact representation of syntactic rules of a computer language based on formal production rules, 66

Formal ontology An ontology is a formal, explicit specification of a shared conceptualization. It includes a vocabulary, semantic interconnections and rules of inference for automated reasoning., 99

Formal verification Assessment of correctness by means of formal proving, 90

Formalism Language of which syntax and semantics are given in an unambiguous formal way., 65

General Systems Theory Approach to study the structure and behavior of *systems* based on interdisciplinary modeling and virtual decomposition of *systems* into hierarchically organized, interacting components, 10

General-purpose Programming Language (GPL) Computer language that is designed under the premise to be universal with respect to specification of programs that use all available features of digital computers (or theoretical equivalents), 69

Geodata Data with a spatial reference that can be used to draw a reference to location on the earth' surface, 42

Geographic Information System Software that is used to collect, manage, analyze and display geodata., 41

Hierarchical system A system composed of components, where components themselves may be composed of components, 8

Homogeneous system A system where subsystems follow the same set of rules, use the same communication pattern and act synchronously., 111

HPP-model The Hardy, de Pazzis and Pomeau CA model for of lattice gas based on the idea of colliding particles on a square lattice with conservation of mass and momentum, 125

Implementation-level tool Modeling tool for EMS that supports the specification of single models, typically associated with the notion of modeling frameworks, 81

Integrated Development Environment A language tool where different functionalities are tightly integrated and accessed via one user interface, 63

Integrated modeling framework Modeling framework that allows the specification of model components according to different modeling paradigms, 81

Interface Formal definition of the outside view of an object or abstraction (e.g. class, module)., 22

Internal DSL DSL where the language concepts are provided by means of extension mechanisms of a host computer language., 72

Knowledge The recognition of something known (e.g. a physical law), or assumed to be known (e.g. a hypothesis), in something new (e.g. an perceived phenomenon), where the known serves as an explanation for the unknown, 50

Language Means to express informational content., 65

Language tool A language tool is software that facilitates humans the production of valid specifications according to a computer language and to realize the informational content of specifications by means of computation (e.g. to execute a calculation), 63

Lattice Boltzmann Method CA that incorporates averages particle motions, 124

Lattice Gas Automaton , 124

Limit cycle Attracting set of periodic points a Dynamical System. The trajectory might actually reach a limit or approached iteratively, 17

Limit point A state $q \in Q$ that a Dynamical System approaches as time goes to infinity. A limit point might be actually reached by a trajectory (stable node) or approached iteratively but never actually reached (stable focus)., 17

Limit set Set of limit points of a given Dynamical System. Here only limit sets wit respect to forward evolution of time is considered, typically referred to as ω-limit set., 17

Mathematical model A mathematical model is a description of a system using mathematical language., 61

Mental model A representation of a model in the mind of the modeler, 10

Mental representation See *mental model*, 10

Meta-metamodel Model of the language that is used to create the language metamodels, 93

Metamodel A model of a modeling language., 92

Metamodeling language A language that is used to specify language metamodels., 98

Metamodeling tool Tool for the specification and management of language metamodels and respective computer languages, 93

Model Simplified conceptualization of a *system* with the purpose of enabling the derivation of useful statements about the system using accessible model representation(s)., 10

Model checking Assessment of the adherence of a model to predefined properties, 90

Model component *Software component* that provides functionality of a simulator as a representation of a model, 77

Model implementation A *model specification* that is processed by a computer in order to automatically set up a *simulator* and execute the simulation, 48

Model representation Accessible incarnation of a model for the purpose of communication and experimentation, 10

Model specification A formal textual or graphical representation of a model, 10

Model transformation The production of another model (target model) of the same system and respective model specification at the base of a specification of a source model. A model transformation is of type model-to-model or model-to-text., 87

Model-based reasoning Model-based reasoning is scientific reasoning, where models are the basic entities to which operations of reasoning are applied., 54

Model-driven Engineering An approach to software and systems engineering with model, transformation between system models and code generation as basic constituting concepts and their specifications as central artefacts., 87

Model-to-model transformation Model transformation that is specified at the base of the meta-model(s) of the languages of the source and the target model., 88

Model-to-text transformation Model transformation that is specified at the base of the meta-model of the language of the source model only and that produces text., 88

Modeling and Simulation (M&S) Method for the characterization of systems based on models and computer simulation., 6

modeling and simulation study A set of tasks with the goal meet respective objectives by means of systems analysis based on system(s), model(s), simulator(s) and experiment(s) with given limited resources., 45

Modeling environment Modeling tool that provides a high-level user interface for the combination of model components and the execution of respective simulations, 81

Modeling framework Modeling tool that allows the specification of models as model components by means of built in abstraction mechanisms (e.g. interfaces) of an programming languages, 81

Modeling language Computer language associated with the specification of models, 69

Modeling paradigm A thought pattern that provides the framework (requirements) for conceptualization of models., 28

Modeling tool Language tool that provides computer languages for the purpose of experimentation., 78

Modeling-level tool Modeling tool for EMS that supports the combination of several model components, 81

Multi-formalism modeling Multi-paradigm modeling, 78

Multi-modeling Multi-paradigm modeling, 78

Multi-paradigm modeling An approach to M&S, where several DSLs that incorporate different modeling paradigms are used for model specification, 78

Non-deterministic system Dynamical system which may produce different behavior under the same conditions, 27

Nucleation Process in a phase transition of a system where the new phase appears localized within the old phase (e.g. gas bubbles in liquid), 124

Object Tangible entity existing in space and time with which one can interact. An object has a well defined state, behavior and an identity, 18

Object-orientation Approach to software engineering, based on the notion of systems being composed of interacting objects that are conceptualized by means of a class hierarchies, 18

Ontology Formal ontology, 99

Open system A system with relevant interaction with the system environment., 8

Orbit Trajectory, 16

Output trajectory Time ordered sequence of outputs of a dynamical system, 49

Percolation Movement of a fluid through a medium that is thought as consisting of randomly connected sites, 124

Periodic point State of a Dynamical System such that there is a period τ such that $\delta^{tau}(q) = q$, 17

Periodic trajectory Trajectory that that is a closed loop, where a single state is reached with period τ. Periodic trajectories can be stable or unstable (cf. limit points), 17

Pragmatics Pragmatics of computer languages is the relation of languages and the context of their usage, 68

Precipitation Formation of a solid through chemical reaction in a solution, 124

Process abstraction Conceptual framework that encourages the decomposition of specification of software by means of abstraction that is based on groupings of statements that describe computations (e.g. assignments, functions, procedures), 70

Programming Environment A collection of tools that makes up a language tool, 63

Programming language Computer language associated with specification of computer programs, 69

Programming language paradigm The set of key abstractions of a programming language., 69

Real system A conceptualization through which a part of reality is perceived as separated from the system's environment by the *system boundary*, 7

Regularity The characteristic of a computer language that it can be used without the need recognize a number of exceptions of the rules of the computer language and without surprises with respect to behavior resulting from unanticipated interaction of language concepts., 71

Scientific workflow Computational workflow that aims at the automatic execution of simulation experiments and experiments series with digital simulators., 77

Self-organized criticality System of which several characterizing quantities have power law behavior, 124

Semantics The semantics of a computer language define the meaning of words uttered in a computer language. Semantics consists of *Static semantics* and *Dynamic semantics.*, 65

Simulation model Model that mimics the behavior of a system, 6

Simulator A mechanical representation of a model used for generating the dynamical behavior of a model by means of executing the instructions of a *model implementation*, 10

Software component Software unit that provides functionality that is to be accessed via a *component platform* according to the interface of a component, 77

Stable fixed point Fixed point where small perturbation remains small (Lyapunov-stability) or where Dynamical System converges to fixed point after perturbation (asymptotically stable), 17

State of a system A characterization of a system at a particular time., 9

State trajectory Time ordered sequence of states of a dynamical system, 49

State variables Subset of all descriptive variables of a Dynamical System that uniquely determines the state of all descriptive variables., 12

Static semantics Static semantics of a computer language is the set of non-syntactical constraints that restrict the set of syntactically valid utterances of a language to a subset., 65

Stationary point $q^* \in Q$ such that $\frac{\partial q}{\partial t} = 0$, 17

Sub-system System element that is itself a system, 9

Super-formalism A formalism that is a combination of a number of formalisms., 79

Supersaturation A solution that contains more of the dissolved material that can be dissolved by the solvent, 133

Syntax The syntax of a computer language is the set of symbols and all valid combinations of symbols of the computer language. Syntax consists of *abstract syntax* and *concrete syntax.*, 65

System See *real system*, 7

System boundary Separates the *system* and its environment, 7

System element Perceived basic entity of a system, 9

Systems analysis The investigation of behavioral aspects of systems, 46

Systems design The investigation of alternative hypothetical future structures of systems, 46

Systems inference The discovery of unknown structural and causal features of systems, 46

Systems Theory See *General Systems Theory*, 10

Testing Assessment of correctness by means of tests (inputs and outputs) that are applied to a system or a model, 90

Time-driven system Dynamical System that is perceived as changing its state at discrete predefined equidistant points in time, 27

Transparency Explicitness of informational content, 7

Type Element of a type hierarchy that prescribes a number of properties and invariant relationships among these properties for all subtypes, 52

Type hierarchy Hierarchy of abstractions (type) that are related by means of operations of model-based reasoning., 52

Uncertainty Any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system, 23

Universality The universality of a computer language refers to the capability of a language to allow the specification of solutions that are specific to a problem domain, 69

Unstable fixed point Fixed point that is not a stable fixed point, 17

Validation (of a model) Process of evaluating and implementing the validity of a *model*, 48

Validity (of a model) Property of a model that it represents its *target system* adequately with respect to the purpose of the model, 48

Verification (of a simulator) Process of evaluating and implementing the correctness of a *simulator*, 48

Bibliography

Homepage: ArcGIS Produkte, March 2012. URL http://esri.de/products/index.html#arcgis.

Homepage: Eclipse Modeling Framework Project (EMF), August 2012a. URL http://www.eclipse.org/modeling/emf/.

Homepage: Implementing Model Integrity in EMF with MDT OCL, August 2012b. URL http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html.

Homepage: ERDAS IMAGINE Developers' Toolkit, March 2012. URL http://www.erdas.com/products/ERDASIMAGINE/IMAGINEDevelopersToolkit/Details.aspx.

Homepage: EMF Documentation, August 2012. URL http://download.eclipse.org/modeling/emf/emf/javadoc/2.8.0/org/eclipse/emf/ecore/package-summary.html#details.

Homepage: Graphical Modeling Project (GMP), August 2012. URL http://www.eclipse.org/modeling/gmp/.

Homepage: GRASS, March 2012. URL http://grass.fbk.eu/intro/general.php.

Homepage: ISO/TC 211 Geographic information/Geomatics, May 2012. URL http://www.isotc211.org/.

Homepage: The Kepler Project, March 2012. URL https://kepler-project.org/.

Hompage: MATLAB Product homepage, March 2012. URL http://www.mathworks.de/help/techdoc/.

Homepage: ModelBuilder, March 2012. URL http://www.esri-germany.de/products/arcgis/about/modelbuilder.html.

Homepage: Open Geospatial Consortium, May 2012. URL http://www.opengeospatial.org/.

Homepage: Open Source GIS, March 2012. URL http://opensourcegis.org/.

Homepage: MMT/QVT Declarative (QVTd), August 2012. URL http://wiki.eclipse.org/QVTd.

Homepage: Project Gigalopolis, 10 2012. URL http://www.ncgia.ucsb.edu/projects/gig/About/about.html.

Homepage: Taverna Workflow Management System, March 2012. URL http://www.taverna.org.uk/.

Homepage: The Triana Project, March 2012. URL `http://www.trianacode.org/`.

A. Alexandridis, D. Vakalis, C.I. Siettos, and G.V. Bafas. A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990. *Applied Mathematics and Computation*, 204(1):191 – 201, 2008. ISSN 0096-3003. doi: DOI:10.1016/j.amc.2008.06.046.

R. M. Argent. An overview of model integration for environmental applications–components, frameworks and semantics. *Environmental Modelling & Software*, 19:219–234, Mar 2004.

R. M. Argent. A case study of environmental modelling and simulation using transplantable components. *Environmental Modelling & Software*, 20:1514–1523, Dec 2005.

R. M. Argent and A. Rizzoli. Development of Multi-Framework Model Components. In C. Pahl-Wostl, S. Schmidt, A.E. Rizzoli, and A.J Jakeman, editors, *Transactions of the Second Biennial Meeting of the International Environmental Modelling and Software Society, Osnabrück, ,Germany*, volume 1, pages 365–370, 2004.

J. L. Aronson, R. Harre, and E. C. Way. *Realism rescued : how scientific progress is possible*. Duckworth, London :, 1994. ISBN 0715624768 0715624768.

C. A. Aumann. Constructing model credibility in the context of policy appraisal. *Environmental Modelling & Software*, 26(3):258 – 265, 2011. ISSN 1364-8152. doi: DOI:10.1016/j.envsoft.2009.09.006. Thematic issue on the assessment and evaluation of environmental models and software.

M. V. Avolio, V. Lupiano, P. Mazzanti, and S. Di Gregorio. A Cellular Automata Model for Flow-like Landslides with Numerical Simulations of Subaerial and Subaqueous Cases. In *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools*. Shaker Verlag, 2009.

R. Axelrod. Advancing the Art of Simulation in the Social Sciences. *Japanese Journal for Management Information System*, 12(3), December 2003.

J. Bai and C. Sander. Lotka-Volterra Modelling, May 2012. URL `http://openwetware.org/wiki/IGEM:IMPERIAL/2006/project/Oscillator/Modelling/LV`.

P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of the 1/f noise. *Phys. Rev. Lett.*, 59:381–384, Jul 1987. doi: 10.1103/PhysRevLett.59.381. URL `http://link.aps.org/doi/10.1103/PhysRevLett.59.381`.

S. Bandini and G. Mauri. Multilayered cellular automata. *Theoretical Computer Science*, 217(1):99 – 113, 1999. ISSN 0304-3975. doi: DOI:10.1016/S0304-3975(98)00152-2.

S. Bandini, G. Mauri, and R. Serra. Cellular automata: From a theoretical parallel computational model to its application to complex systems. *Parallel Computing*, 27(5): 539 – 553, 2001. ISSN 0167-8191. doi: DOI:10.1016/S0167-8191(00)00076-4.

J. Banks. Simulation fundamentals: simulation fundamentals. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 9–16, San Diego, CA, USA, 2000. Society for Computer Simulation International. ISBN 0-7803-6582-8.

I. Benenson and P. M. Torrens. *System Theory, Geography, and Urban Modeling*. John Wiley & Sons, Ltd, 2006a.

I. Benenson and P. M. Torrens. *Formalizing Geosimulation with Geographic Automata Systems (GAS)*. John Wiley & Sons, Ltd, 2006b. URL `http://dx.doi.org/10.1002/0470020997.ch2`.

L. v. Bertalanffy. An Outline of General System Theory. *The British Journal for the Philosophy of Science*, 1(2):134–165, 1950. ISSN 00070882.

K. Beven. Towards a coherent philosophy for modelling the environment. *Proceedings of the Royal Society A*, 458:2465–2484, 2002.

J. Bezivin and I. Kurtev. Model-based Technology Integration with the Technical Space Concept. In *Metainformatics Symposium 2005*, Esbjerg, Denmark, 2005.

F. Blanchard. Topological chaos: what may this mean? *Journal of Difference Equations and Applications*, 15(1):23–46, 2009. doi: 10.1080/10236190802385355.

N. Boccara. *Modeling Complex Systems*. Graduate texts in contemporary physics. Springer-Verlag New York, Inc., 2004.

J. J. Boersema. Environmental Sciences, Sutainability, and Quality. In Jan J. Boersema and Lucas Reijnders, editors, *Principles of Environmental Sciences*, chapter 1, pages 3 – 14. Springer Science Business + Media B.V., 2009.

G. Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004. ISBN 020189551X.

A. Borshchev and A. Filippov. From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. *22nd International Conference of the System Dynamics Society*, page 45, 2004.

H. Bossel. *Systeme, Dynamik, Simulation. Modellbildung, Analyse und Simulation komplexer Systeme*. Books on Demand GmbH, 1 edition, 2004. ISBN 3833409843.

M. Brugnach, C. Pahl-Wostl, K.E. Lindenschmidt, J.A.E.B. Janssen, T. Filatova, A. Mouton, G. Holtz, P. van der Keur, and N. Gaber. Chapter Four Complexity and Uncertainty: Rethinking the Modelling Activity. In A.E. Rizzoli A.J. Jakeman, A.A. Voinov and S.H. Chen, editors, *Environmental Modelling, Software and Decision Support*, volume 3 of *Developments in Integrated Environmental Assessment*, pages 49 – 68. Elsevier, 2008. doi: DOI:10.1016/S1574-101X(08)00604-2.

I. Ceh, M. Crepinsek, T. Kosar, and M. Mernik. Ontology driven development of domain-specific languages. *Comput. Sci. Inf. Syst.*, pages 317–342, 2011.

H. Chaté and P. Manneville. Criticality in cellular automata. *Phys. D*, 45(1-3):122–135, October 1990. ISSN 0167-2789. doi: 10.1016/0167-2789(90)90178-R.

S. Chen, K. Diemer, G. D. Doolen, K. Eggert, C. Fu, S. Gutman, and B. J. Travis. Lattice gas automata for flow through porous media. *Physica D: Nonlinear Phenomena*, 47(1–2): 72 – 84, 1991. ISSN 0167-2789. doi: 10.1016/0167-2789(91)90281-D.

B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Aléa-Saclay. Cambridge University Press, 1998. ISBN 9780521673457.

D. Chorafas. *Systems and Simulation*, volume 14 of *Mathematics in Science and Engineering*. ACADEMIC PRESS INC, New York, 1965.

M. Cook. Universality in Elementary Cellular Automata. *Complex Systems*, 15(1):1–40, 2004.

R. Costanza and A. Voinov. Modeling ecological and economic systems with STELLA: Part III. *Ecological Modelling*, 143(1-2):1 – 7, 2001. ISSN 0304-3800. doi: DOI:10.1016/S0304-3800(01)00358-1.

H. Couclelis. Cellular worlds: a framework for modeling micro - macro dynamics. *Environment and Planning A*, 17(5):585 – 596, 1985.

M. Creutz. Cellular Automata And Self Organized Criticality. In *in Some New Directions in Science on Computers*, 1996.

K. Culik, II and S. Yu. Undecidability of CA classification schemes. *Complex Syst.*, 2(2): 177–190, 1988. ISSN 0891-2513.

R. A. d. By, R. A. Knippers, Y. Sun, M. C. Ellis, Menno-Jan Kraak, Michael J. C. Weir, Yola Georgiadou, Mostafa M. Radwan, Cees J. van Westen, Wolfgang Kainz, and Edmund J. Sides. *Principles of Geographic Information Systems*. ITC Educational Textbook Series. The International Institute for Aerospace Survey and Earth Sciences (ITC), 2000.

J. d. Lara and H. Vangheluwe. ATOM 3 : A Tool for Multi-Formalism Modelling and Meta-Modelling. 2002.

D. Dab, J.-P. Boon, and Y.-X. Li. Lattice-gas automata for coupled reaction-diffusion equations. *Phys. Rev. Lett.*, 66:2535–2538, May 1991. doi: 10.1103/PhysRevLett.66.2535.

J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. The Department of Defense High Level Architecture. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 142–149, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-7803-4278-X. doi: http://doi.acm.org/10.1145/268437.268465.

O. David, I. W. Schneider, and G. H. Leavesley. Metadata and Modeling Frameworks: The Object Modeling System Example. In *Trans. 2nd Bienn. Meeting of the Int. Environ. Modelling and Software Soc., iEMSs 2004*, 2004.

J. F. Derry. Modelling ecological interaction despite object-oriented modularity. *Ecological Modelling*, 107(2-3):145 – 158, 1998. ISSN 0304-3800. doi: DOI:10.1016/S0304-3800(97)00214-7.

Robert L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, Redwood City, CA, 2nd edition, 1989.

A.K. Dewdney. Cellular Automata. In Sven Erik Jorgensen and Brian Fath, editors, *Encyclopedia of Ecology*, pages 541 – 550. Academic Press, Oxford, 2008. ISBN 978-0-08-045405-4. doi: DOI:10.1016/B978-008045405-4.00147-6.

C. Dilworth. *Principles of Environmental Sciences*, chapter General Principles, pages 75 – 84. Springer Science + Business Media B.V., 2009.

R.A. Dow. Additive cellular automata and global injectivity. *Physica D: Nonlinear Phenomena*, 110(1–2):67 – 91, 1997. ISSN 0167-2789. doi: 10.1016/S0167-2789(97)00074-2.

J. K. Doyle and D. N. Ford. Mental models concepts for system dynamics research. *System Dynamics Review*, 14(1):3–29, 1998. ISSN 1099-1727. doi: 10.1002/(SICI)1099-1727(199821)14:1\%3C3::AID-SDR140\%3E3.0.CO;2-K.

A. Drogoul, D. Vanbergue, and T. Meurisse. *Multi-agent Based Simulation: Where Are the Agents?*, chapter Multi-agent Based Simulation: Where Are the Agents?, pages 43–49. 2003. 10.1007/3-540-36483-8_1.

B. Drossel and F. Schwabl. Self-organized critical forest-fire model. *Physical Review Letters*, 69(11), Sep 1992. doi: 10.1103/PhysRevLett.69.1629.

C. Duchêne and J. Gaffuri. Combining Three Multi-agent Based Generalisation Models: AGENT, CartACom and GAEL. In Anne Ruas, Christopher Gold, William Cartwright, Georg Gartner, Liqiu Meng, and Michael P. Peterson, editors, *Headway in Spatial Data Handling*, Lecture Notes in Geoinformation and Cartography, pages 277–296. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-68566-1.

K. Dunbar. The Scientist InVivo: How scientists think and reason in the laboratory. In L. Magnani, N. Nersessian, and P. Thagard, editors, *Model-based reasoning in scientific discovery*, pages 89 – 98. Plenum Press, 1999.

B. Durand, E. Formenti, and Z. Róka. Number-conserving cellular automata I: decidability. *Theoretical Computer Science*, 299(1–3):523 – 535, 2003. ISSN 0304-3975. doi: 10.1016/S0304-3975(02)00534-0.

R. Dvorak. Model Transformation with Operational QVT. http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf, 2008.

J. W. Eaton, D. Bateman, and S. Hauberg. *GNU Octave - A high-level interactive language for numerical computations*, 3 edition, February 2011.

S. Efftinge, P. Friese, A. Haase, C. Kadura, Bernd Kolb, Dieter Moroff, Karsten Thoms, and Markus Völter. *openarchitectureWare User Guide Version 4.2*, 2007.

M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In *in The 6th OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2006*, pages 123–139. ACM, ACM Press, 2006.

A. Fall and J. Fall. A domain-specific language for models of landscape dynamics. *Ecological Modelling*, 141(1-3):1–18, July 2001.

M. Feilkas. How to represent Models, Languages and Transformations? In J. Gray, J.-P. Tolvanen, and J. Sprinkle, editors, *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, Computer Science and Information System Reports, Technical Reports, TR-37. University of Jyväskylä, 2006.

J-B. Filippi and P. Bisgambiglia. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. *Environmental Modelling & Software*, 19(3): 261–274, March 2004.

R. A. Finkel. *Advanced programming language design*. Addison-Wesley Publishing Company, 1996. ISBN 978-0-201-06824-5.

J. Fischer and K. Ahrens. *Objektorientierte Prozeßsimulation in C++*. Addison-Wesley, 1996.

J. Fischer, M. Piefel, and M. Scheidgen. A Metamodel for SDL-2000 in the Context of Metamodelling ULF. In Daniel Amyot and Alan W. Williams, editors, *SAM*, volume 3319 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2004. ISBN 3-540-24561-8.

F. Fondement and R. Silaghi. Defining Model Driven Engineering Processes. In *Proceedings of the 3rd Workshop in Software Model Engineering (WiSME'04)*, 2004.

A. Ford. *Modeling the environment*. Island Press, 2009. ISBN 9781597264730.

E. Fredkin. An Introduction to Digital Philosophy. *International Journal of Theoretical Physics*, 42(2):189–247, 2003.

R. Frigg. Self-organised criticality—what it is and what it isn't. *Studies in History and Philosophy of Science Part A*, 34(3):613 – 632, 2003. ISSN 0039-3681. doi: 10.1016/S0039-3681(03)00046-3.

R. Frigg. Scientific Representation and the Semantic View of Theories. *THEORIA. An International Journal for Theory, History and Foundations of Science*, 21(1), 2006. ISSN 2171-679X.

R. Frigg and S. Hartmann. Models in Science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2009 edition, 2009.

U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-Gas Automata for the Navier-Stokes Equation. *Physical Review Letters*, 56(14):1505–1508, April 1986. doi: 10.1103/PhysRevLett.56.1505.

U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet. Lattice gas hydrodynamics in two and three dimensions. *Complex Syst.*, 1:649–707, 1987.

P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Computer Society Pr, 2003. ISBN 0471471631.

P. Gabriel, M. Goulão, and V. Amaral. Do Software Languages Engineers Evaluate their Languages? In *Proceedings of the XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010)*. Universidad del Azuay, Cuenca, Ecuador, 2010.

M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi. *GNU Scientific Library - Reference Manual*, 1.15 edition, 2011.

M. Gardner. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, pages 120–123, October 1970.

Dragan Gasevic, Dragan Djuric, Vladan Devedzic, and Bran Selic. *Model Driven Architecture and Ontology Development*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540321802.

R. N. Giere. The Cognitive Structure of Scientific Theories. *Philosophy of Science*, 61(2): pp. 276–296, 1994. ISSN 00318248.

U. Golze. (A-)synchronous (non-)deterministic cell spaces simulating each other. *Journal of Computer and System Sciences*, 17(2):176 – 193, 1978. ISSN 0022-0000. doi: DOI: 10.1016/0022-0000(78)90003-X.

S. Di Gregorio and R. Serra. An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Future Generation Computer Systems*, 16(2-3):259 – 271, 1999. ISSN 0167-739X. doi: DOI:10.1016/S0167-739X(99)00051-5.

S. Di Gregorio, R. Serra, and M. Villani. Applying cellular automata to complex environmental problems: The simulation of the bioremediation of contaminated soils. *Theoretical Computer Science*, 217(1):131 – 156, 1999. ISSN 0304-3975. doi: DOI: 10.1016/S0304-3975(98)00154-6.

C. Grelck, F. Penczek, and K. Trojahner. CAOS: A Domain-Specific Language for the Parallel Simulation of Cellular Automata. In Victor Malyshkin, editor, *Parallel Computing Technologies*, volume 4671 of *Lecture Notes in Computer Science*, pages 410–417. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-73939-5.

G. Guizzardi, L. F. Pires, and M. V. Sinderen. An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling. In *Languages, ACM/IEEE 8 th International Conference on Model Driven Engineering Languages and Systems, Montego*, pages 691–705, 2005.

H. A. Gutowitz. A hierarchical classification of cellular automata. *Physica D: Nonlinear Phenomena*, 45(1-3):136 – 156, 1990. ISSN 0167-2789. doi: DOI:10.1016/0167-2789(90)90179-S.

C. Hardebolle and F. Boulanger. Exploring Multi-Paradigm Modeling Techniques. *Simulation*, 85(11-12):688–708, 2009. ISSN 0037-5497. doi: http://dx.doi.org/10.1177/0037549709105240.

D. Harel and B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff - Part I: The Basic Stuff. Technical Report MCS00-16, 2000.

D. Harel and B. Rumpe. Meaningful modeling: What's the semantics of "semantics"? *COMPUTER*, 37(10):64–72, 2004. ISSN 0018-9162. doi: http://doi.ieeecomputersociety.org/10.1109/MC.2004.172.

E. Harmon and N. J. Nersessian. Cognitive partnerships on the bench top: designing to support scientific researchers. In Johann van der Schijff and Gary Marsden, editors, *Conference on Designing Interactive Systems*, pages 119–128. ACM, 2008. ISBN 978-1-60558-002-9.

R. Harré. *Modeling: Gateway to the Unknown*, volume 1 of *Studies in Multidisciplinarity*. Elsevier, 2004.

M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978. ISBN 0201029553.

O. Haugen and P. Mohagheghi. A Multi-dimensional Framework for Characterizing Domain Specific Languages. In J. Sprinkle, J. Gray, M. Rossi, and J.-P. Tolvanen, editors, *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07)*, number TR-38 in Computer Science and Information System Reports, Technical Reports. University of Jyväskylä, Finland, 2007.

K. Helsgaun. jDisco - a java package for combined discrete event and continous simulation. Technical report, Department of Computer Science, Roskilde University, 2001.

J. H. Hill. Measuring and Reducing Modeling Effort in Domain-Specific Modeling Languages with Examples. In *Proceedings of the 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, ECBS '11, pages 120–129, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4379-6. doi: 10.1109/ECBS.2011.22.

C. Hillyer, J. Bolte, F. v. Evert, and A. Lamaker. The ModCom modular simulation system. *European Journal of Agronomy*, 18(3-4):333 – 343, 2003. ISSN 1161-0301. doi: DOI:10.1016/S1161-0301(02)00111-9. Modelling Cropping Systems: Science, Software and Applications.

B. Hjørland. Domain Analysis in Information Science. In *Encyclopedia of Library and Information Science*, pages 1 – 7. Taylor & Francis, second edition edition, 2004.

P. Hogeweg. Cellular automata as a paradigm for ecological modeling. *Applied Mathematics and Computation*, 27(1):81 – 100, 1988. ISSN 0096-3003. doi: DOI:10.1016/0096-3003(88)90100-2.

E. Holz. *Kombination von Modellierungstechniken für den Softwareentwurf*. Der Andere Verlag, 2004.

D. Holzworth and N. Huth. Reflection + XML Simplifies Development of the APSIM Generic PLANT Model. In R.S. Anderssen, Braddock R.D., and L.T.H. Newham, editors, *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, 2009.

D. P. Holzworth, N. I. Huth, and P. G. d. Voil. Simplifying environmental model reuse. *Environmental Modelling & Software*, In Press, Corrected Proof, 2008. ISSN 1364-8152. doi: DOI:10.1016/j.envsoft.2008.10.018.

R. I. G. Hughes. Models and Representation. *Philosophy of Science*, 64:pp. S325–S336, 1997. ISSN 00318248.

J. R. Hummel and J. H Christiansen. The dynamic information architecture system: a simulation framework to provide interoperability for process models. In *Proceedings of Simulation Interoperability Workshop, Orlando, FL, 8-13th September*, 2002.

M. H. Hwang. *Modeling and Simulation using DEVS#*. http://xsy-csharp.sourceforge.net/DEVSsharp, first edition, May 2007.

M. H. Hwang. *DEVS++: C++ Open Source Library of DEVS Formalism*. http://odevspp.sourceforge.net/, v.1.4.2 edition, April 2009.

ISO. ISO/DIS 19101 Geographic Information - Reference Model, 2000.

ISO. Geographic information - Methodology for feature cataloguing DRAFT, 2001.

ISO. ISO/DIS 19109 Geographic information - Rules for application DRAFT, 2002.

ISO. ISO/DIS 19107 Geographic information - Spatial Schema DRAFT, 2003a.

ISO. ISO 19115 Geographic information Metadata FINAL DRAFT, 2003b.

R. M. Itami. Simulating spatial dynamics: cellular automata theory. *Landscape and Urban Planning*, 30(1-2):27 – 47, 1994. ISSN 0169-2046. doi: DOI:10.1016/0169-2046(94)90065-5. Special Issue Landscape Planning: Expanding the Tool Kit.

S. Jafer and G. A. Wainer. Parallel Algorithms for Cellular Models Simulation. *Parallel Processing Letters*, 17:263–285, 2007. doi: 10.1142/S0129626407003010.

L. Jahnke and J. W. Kantelhardt. Comparison of models and lattice-gas simulations for Liesegang patterns. *The European Physical Journal - Special Topics*, 161:121–141, 2008. ISSN 1951-6355. 10.1140/epjst/e2008-00755-2.

Anthony J. Jakeman, Alexey A. Voinov, Andrea E. Rizzoli, and Serena H. Chen, editors. *Environmental Modelling, Software and Decision Support*, volume 3 of *Developments in Integrated Environmental Assessment*. Elsevier, 1 edition, 2008.

J. A. Joines and S. D. Roberts. Fundamentals of object-oriented simulation. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 141–150, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. ISBN 0-7803-5134-7.

R. Kapral, A. Lawniczak, and P. Masiar. Oscillations and waves in a reactive lattice-gas automaton. *Phys. Rev. Lett.*, 66:2539–2542, May 1991. doi: 10.1103/PhysRevLett.66.2539.

T. Karapiperis. Cellular Automaton Models of Reaction-Transport Processes. In Ingmar Grenthe and Ignasi Puigdomenech, editors, *MODELLING IN AQUATIC CHEMISTRY*, chapter Chapter XI, pages 495 – 524. OECD Publications, 1997.

J. Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1-3):3 – 33, 2005. ISSN 0304-3975. doi: DOI:10.1016/j.tcs.2004.11.021.

W. J. Karplus. The spectrum of mathematical modeling and systems simulation. *Mathematics and Computers in Simulation*, 19(1):3 – 10, 1977. ISSN 0378-4754. doi: DOI:10.1016/0378-4754(77)90034-9.

G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schneider, and S. Völkel. Design Guidelines for Domain Specific Languages. In Matti Rossi, Jonathan Sprinkle, Jeff Gray, and Juha-Pekka Tolvanen, editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, pages 7–13, 2009.

D. Karssenberg. *Building dynamic spatial environmental models*. PhD thesis, Utrecht University, 2002.

Y. Kayama, M. Tabuse, H. Nishimura, and T. Horiguchi. Characteristic parameters and classification of one-dimensional cellular automata. *Chaos, Solitons & Fractals*, 3 (6):651 – 665, 1993. ISSN 0960-0779. doi: 10.1016/0960-0779(93)90051-2.

R. M. Keller and J. L. Dungan. Meta-modeling: a knowledge-based approach to facilitating process model construction and reuse. *Ecological Modelling*, 119(2-3):89 – 116, 1999. ISSN 0304-3800. doi: DOI:10.1016/S0304-3800(98)00197-5.

W.G. Kelley and A.C. Peterson. *Difference Equations: An Introduction With Applications*. Acad. Press, 2001. ISBN 9780124033306.

S. Kent. Model Driven Engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 286–298, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43703-7.

H. Kern, A. Hummel, and S. Kühne. Towards a comparative analysis of meta-metamodels. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 7–12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1183-0. doi: 10.1145/2095050.2095053.

F. Kühnlenz, F. Theisselmann, and J. Fischer. Model-driven Engineering for Transparent Environmental Modeling and Simulation. In F. Breitenecker I. Troch, editor, *Proceedings MATHMOD 09 Vienna*, number 35 in ARGESIM-Reports, 2009.

D. Kirsh. When is information explicitly represented? In *The Vancouver Studies in Cognitive Science.*, pages 340–365. Oxford University Press, 1990.

D. Kirsh. Implicit and Explicit Representation. In Lynn Nadel, editor, *Encyclopedia of Cognitive Science*, volume 2, pages 478–481. Wiley, 2005.

D. Kirsh. Thinking with external representations. *AI & Society*, 25:441–454, 2010. ISSN 0951-5666. 10.1007/s00146-010-0272-8.

D. Klahr. Searching for cognition in cognitive models of science. *PSYCOLOQUY*, 5(69), 1994.

J. P. C. Kleijnen. Verification and validation of simulation models. *European Journal of Operational Research*, 82(1):145 – 162, 1995. ISSN 0377-2217. doi: DOI:10.1016/0377-2217(94)00016-6.

J.P.C. Kleijnen. Experimental Design for Sensitivity Analysis, Optimization and Validation of Simulation Models. Discussion Paper 1997-52, Tilburg University, Center for Economic Research, 1997.

G. J. Klir. *Architecture of Systems Problem Solving*. International Federation for Systems Research international series on systems science and engineering. Kluwer Academic Press / Plenum Publishers, 2nd edition, 2003. ISBN 0306473577.

K. Kobayashi and D. Goldstein. On formulations of firing squad synchronization problems. In *Proceedings of the 4th international conference on Unconventional Computation*, UC'05, pages 157–168, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29100-8, 978-3-540-29100-8. doi: 10.1007/11560319_15.

H. Krahn, B. Rumpe, and S. Völkel. MontiCore: Modular Development of Textual Domain Specific Languages. In R. F. Paige and Bertrand M., editors, *TOOLS (46)*, volume 11 of *Lecture Notes in Business Information Processing*, pages 297–315. Springer, 2008. ISBN 978-3-540-69823-4.

S. Kralisch and P. Krause. JAMS - A Framework for Natural Resource Model Development and Application. In Voinov A., Jakeman A., and Rizzoli A., editors, *Proceedings of the iEMSs Third Biannual Meeting 'Summit on Environmental Modelling and Software'*, Burlington, USA, 2006.

S. Kralisch, P. Krause, and O. David. Using the object modeling system for hydrological model development and application. *Advances In Geosciences*, 4:75–81, 2005.

A. Kunert. *Prozessorientierte optimistisch-parallele Simulation.* PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät II der Humboldt-Universität zu Berlin, 2010.

A. Laarman and I. Kurtev. Ontological Metamodeling with Explicit Instantiation. In *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 174–183, Heidelberg, January 2010. Springer Verlag.

C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1-3):12 – 37, 1990. ISSN 0167-2789. doi: DOI:10.1016/0167-2789(90)90064-V.

G. H. Leavesley, S.L. Markstrom, M.S. Brewer, and R.J. Viger. The Modular Modeling System (MMS) - The physical process modeling component of a database-centered decision support system for water and power management. *Water, Air, & Soil Pollution*, 90(1-2):303–311, 1996a.

G.H. Leavesley, P.J. Restrepo, L.G. Stannard, L.A. Frankowski, and A.M. Sautins. *The Modular Modeling System (MMS): a modeling framework for multidisciplinary research and operational applications*, chapter GIS and Environmental Modeling: Progress and Research Issues, page 155?158. World Books, Ft. Collins, CO., 1996b.

D. B. Lee. Requiem for large-scale models. *SIGSIM Simul. Dig.*, 6:16–29, January 1975. ISSN 0163-6103. doi: http://doi.acm.org/10.1145/1102945.1102950.

W. Li, N H. Packard, and C. G. Langton. Transition phenomena in cellular automata rule space. *Physica D: Nonlinear Phenomena*, 45(1–3):77 – 94, 1990. ISSN 0167-2789. doi: 10.1016/0167-2789(90)90175-O.

Q. Liu. *Algorithms for Parallel Simulation of Large-Scale DEVS and Cell-DEVS Models.* PhD thesis, Systems and Computer Engineering Dep. Carleton University, 1125 Colonel By Dr. Ottawa, ON, Canada K1S 5B6, Sep 2010.

K.C. Louden. *Programming languages: principles and practice.* Computer Science Series. Brooks/Cole, 2003. ISBN 9780534953416.

B. Ludäscher, M. Weske, T. Mcphillips, and S. Bowers. Scientific Workflows: Business as Usual? In *Proceedings of the 7th International Conference on Business Process Management*, BPM '09, pages 31–47, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03847-1. doi: 10.1007/978-3-642-03848-8_4.

K.L. Man, T. Krilavicius, and K. Wan. Recent Advanced Languages and Tools for Hybrid Systems. *IAENG International Journal of Computer Science*, 37(3), 2010.

S. M. Manson. Simplifying complexity: a review of complexity theory. *Geoforum*, 32(3): 405 – 414, 2001. ISSN 0016-7185. doi: DOI:10.1016/S0016-7185(00)00035-X.

G. Manzini and L. Margara. Attractors of Linear Cellular Automata. *Journal of Computer and System Sciences*, 58(3):597 – 610, 1999. ISSN 0022-0000. doi: 10.1006/jcss.1998.1609.

B. Marchionni and D. Ames. A Modular Spatial Modeling Environment for GIS. In Tyler Mitchell, editor, *FOSS4G 2009 Conference Proceedings*, pages 53 – 61. OSGeo, 2009.

L. Margara. On Some Topological Properties of Linear Cellular Automata. In MirosÅ,aw KutyÅ,owski, Leszek Pacholski, and Tomasz Wierzbicki, editors, *Mathematical Foundations of Computer Science 1999*, volume 1672 of *Lecture Notes in Computer Science*, pages 209–219. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66408-6. doi: 10.1007/3-540-48340-3_19.

N. Margolus. Physics-like Models of Computation. *Physica*, 10D:81 – 95, 1984.

T. Maxwell and R. Costanza. A language for modular spatio-temporal simulation. *Ecological Modelling*, 103(2-3):105 – 113, 1997. ISSN 0304-3800. doi: DOI:10.1016/S0304-3800(97)00103-8.

S. Mazzoleni, F. Giannino, M. Mulligan, D. Heathfield, M. Colandrea, M. Nicolazzo, and M. d'Aquino. A new raster-based spatial modelling system: 5d environment. In A. Voinov, A.J. Jakeman, and A.E. Rizzoli, editors, *Proceedings of the iEMSs Third Biennial Meeting: "Summit on Environmental Modelling and Software". International Environmental Modelling and Software Society, Burlington, USA*, 2006.

B.S. McIntosh, C. Giupponi, A.A. Voinov, C. Smith, K.B. Matthews, M. Monticino, M.J. Kolkman, N. Crossman, M. van Ittersum, D. Haase, A. Haase, J. Mysiak, J.C.J. Groot, S. Sieber, P. Verweij, N. Quinn, P. Waeger, N. Gaber, D. Hepting, H. Scholten, A. Sulis, H. van Delden, E. Gaddis, and H. Assaf. Chapter Three Bridging the Gaps Between Design and Use: Developing Tools to Support Environmental Management and Policy. In A.E. Rizzoli A.J. Jakeman, A.A. Voinov and S.H. Chen, editors, *Environmental Modelling, Software and Decision Support*, volume 3 of *Developments in Integrated Environmental Assessment*, pages 33 – 48. Elsevier, 2008. doi: DOI:10.1016/S1574-101X(08)00603-0.

M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1118890.1118892.

J. Miller and J. Mukerji. MDA Guide Version 1.0.1, 2003.

M. Mitchell. Computation in Cellular Automata: A Selected Review. pages 95–140, 1996.

A. Müller. Geoinformationssystem. In Jürgen Bollmann and Wolf Günther Koch, editors, *Lexikon der Kartographie und Geomatik*, volume 1, pages 304 – 305. Spektrum Akademischer Verlag, 2001a.

A. Müller. *Lexikon der Kartographie und Geomatik*, volume 1. Spektrum Akademischer Verlag, 2001b.

A. Moreira. Universality and decidability of number-conserving cellular automata. *Theoretical Computer Science*, 292(3):711 – 721, 2003. ISSN 0304-3975. doi: 10.1016/S0304-3975(02)00065-8. <ce:title>Algorithms in Quantum Information Prcoessing</ce:title>.

P. D. Mosses. Formal Semantics of Programming Languages: - An Overview -. *Electronic Notes in Theoretical Computer Science*, 148(1):41 – 73, 2006. ISSN 1571-0661. doi: 10.1016/j.entcs.2005.12.012. <ce:title>Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004)</ce:title> <xocs:full-name>Foundations of Visual Modelling Techniques 2004</xocs:full-name>.

P. J. Mosterman and H. Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. *Simulation*, 80(9):433–450, 2004.

R. Muetzelfeldt and J. Massheder. The Simile visual modelling environment. *Europ. J. Agronomy*, 18:345–358, 2003.

A. Muzy, J.J. Nutaro, B.P. Zeigler, and P. Coquillard. Modeling and simulation of fire spreading through the activity tracking paradigm. *Ecological Modelling*, 219(1-2):212 – 225, 2008. ISSN 0304-3800. doi: DOI:10.1016/j.ecolmodel.2008.08.017.

N. J. Nersessian. Model-based reasoning in conceptual change. In L. Magnani, N. J. Nersessian, and P. Thagard, editors, *Model-Based Reasoning in Scientific Discovery*, pages 5 – 22. Kluwer Academic/Plenum Publishers, New York, 1999.

N. J. Nersessian. The cognitive basis of model-based reasoning in science. In P. Carruthers, S. Stich, and M. Siegal, editors, *The Cognitive Basis of Science*, pages 133 – 153. Cambridge University Press, 2002a.

N. J. Nersessian. Abstraction via generic modeling in concept formation in science. *Mind & Society*, 3:129–154, 2002b. ISSN 1593-7879. 10.1007/BF02511871.

N. J. Nersessian and C. Patton. Model-Based Reasoning in Interdisciplinary Engineering. In Anthonie Meijers, editor, *Philosophy of Technology and Engineering Sciences*, pages 727 – 757. North-Holland, Amsterdam, 2009. ISBN 978-0-444-51667-1. doi: DOI: 10.1016/B978-0-444-51667-1.50031-8.

J. Nutaro. *A Discrete EVent system Simulator*, September 2011. URL `http://www.ornl.gov/~1qn/adevs/adevs-docs/manual.pdf`.

Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. Technical report, November 2007.

OGC. OpenGIS Grid Coverage Service Implementation Specification , 2001.

OGC. OGC Reference Model, 2003a.

OGC. OGC Reference Model. Technical report, Open Geospatial Consortium Inc., 2003b.

OGC. OGC Abstract Specification Topic 2, Spatial referencing by coordinates, 2004a.

OGC. The OGC Abstract Specification Topic 0: Abstract Specification Overview, 2004b.

OGC. OpenGIS Geographic Objects Implementation Specification , 2005.

OGC. The OpenGIS® Abstract Specification Topic 6: Schema for coverage geometry and functions, 2006.

OGC. OGC Reference Model, 2008.

OGC. The OpenGIS Abstract Specification Topic 5: Features, 2009.

A. Ohgai, Y. Gohnai, and K. Watanabe. Cellular automata modeling of fire spread in built-up areas–A tool to aid community-based planning for disaster mitigation. *Computers, Environment and Urban Systems*, 31:441–460, Jul 2007.

omg. *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006. URL `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.

C. M. Overstreet and R. E. Nance. A specification language to assist in analysis of discrete event simulation models. *Commun. ACM*, 28:190–201, February 1985. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/2786.2792.

T. Parr. *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. The Pragmatic Bookshelf, 2009.

F. S. Parreiras, S. Staab, and A. Winter. On marrying ontological and metamodeling technical spaces. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 439–448, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi: 10.1145/1287624.1287687.

M. Pfeiffer and J. Pichler. A Comparison of Tool Support for Textual Domain-Specific Languages. In *The 8th OOPSLA Workshop on Domain-Specific Modeling*, 2008.

M. Piefel. A common metamodel for code generation. In *Proceedings of the 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications. I I I S*, 2006.

R. Popma. JET Tutorial Part 1 (Introduction to JET), August 2012. URL `http://help.eclipse.org/indigo/index.jsp?topic=/org.eclipse.emf.doc/tutorials/jet1/jet_tutorial1.html`.

K. Popper. *The logic of scientific discovery*. Routledge Classics. Routledge, 2002.

E. Posse, A. Muzy, and H. Vangheluwe. A framework for visual specification and simulation of cellular systems. In *Proceedings of the 2006 DEVS Integrative M&S Symposium (DEVS'06)*, 2006.

H. Praehofer, F. Auering, and G. Reisinger. An environment for DEVS-based multi-formalism simulation in common Lisp/CLOS. *Discrete Event Dynamic Systems: Theory and Applications*, 3(2/3):119–149, 1993.

D. Pullar. SimuMap: a computational system for spatial modelling. *Environmental Modelling & Software*, 19:235–243, Mar 2004.

G. Quesnel, R. Duboz, and É. Ramat. The Virtual Laboratory Environment - An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4):641 – 653, 2009. ISSN 1569-190X. doi: DOI:10.1016/j.simpat.2008.11.003.

J. M. Rahman, S. P. Seaton, J.-M. Perraud, H. Hotham, D. I. Verrelli, and J. R. Coleman. It's TIME for a new environmental modelling framework. In D. A. Post, editor, *MODSIM 2003 International Congress on Modelling and Simulation: Townsville, Modelling and Simulation Society of Australia and New Zealand Inc.*, pages 1727–1732, 2003.

J. M. Rahman, S. M. Cuddy, and F. G. R. Watson. Tarsier and ICMS: two approaches to framework development. *Mathematics and Computers in Simulation*, 64(3-4):339 – 350, 2004a. ISSN 0378-4754. doi: DOI:10.1016/S0378-4754(03)00100-9. MSSANZ/IMACS 14th Biennial Confernece on Modelling and Simulation.

J. M. Rahman, S. P. Seaton, and S. M. Cuddy. Making frameworks more useable: using model introspection and metadata to develop model processing tools. *Environmental Modelling & Software*, 19(3):275 – 284, 2004b. ISSN 1364-8152. doi: DOI:10.1016/S1364-8152(03)00153-1. Concepts, Methods and Applications in Environmental Model Integration.

E. Ramat and P. Preux. Virtual laboratory environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems. *Simulation Modelling Practice and Theory*, 11(1):45 – 55, 2003. ISSN 1569-190X. doi: DOI:10.1016/S1569-190X(02)00094-1. Modelling and Simulation: Analysis, Design and Optimisation of Industrial Systems.

G. W. Recktenwald. Finite-Difference Approximations to the Heat Equation, May 2012. URL `http://www.f.kth.se/~jjalap/numme/FDheat.pdf`.

J. C. Refsgaard, J. P. v. d. Sluijs, A. L. Højberg, and P. A. Vanrolleghem. Uncertainty in the environmental modelling process – A framework and guidance. *Environmental Modelling & Software*, 22(11):1543 – 1556, 2007. ISSN 1364-8152. doi: 10.1016/j.envsoft.2007.02.004.

A. E. Rizzoli, M. Donatelli, I. N. Athanasiadis, F. Villa, and D. Huber. Semantic links in integrated modelling frameworks. *Mathematics and Computers in Simulation*, 78(2-3):412 – 423, 2008a. ISSN 0378-4754. doi: DOI:10.1016/j.matcom.2008.01.017. Special Issue: Selected Papers of the MSSANZ/IMACS 16th Biennial Conference on Modelling and Simulation, Melbourne, Australia, 12-15 December 2005.

A.E. Rizzoli, M.G.E. Svensson, E.C. Rowe, M. Donatelli, R. Muetzelfeldt, T. van der Wal, F.K. van Evert, and F. Villa. Modelling Framework (SeamFrame) requirements. SEAMLESS report 6, SEAMLESS, December 2005.

A.E. Rizzoli, G. Leavesley, J.C. Ascough II, R.M. Argent, I.N. Athanasiadis, V. Brilhante, F.H.A. Claeys, O. David, M. Donatelli, P. Gijsbers, D. Havlik, A. Kassahun, P. Krause, N.W.T. Quinn, H. Scholten, R.S. Sojda, and F. Villa. Chapter Seven Integrated Modelling Frameworks for Environmental Assessment and Decision Support. In A.E. Rizzoli

A.J. Jakeman, A.A. Voinov and S.H. Chen, editors, *Environmental Modelling, Software and Decision Support*, volume 3 of *Developments in Integrated Environmental Assessment*, pages 101 – 118. Elsevier, 2008b. doi: DOI:10.1016/S1574-101X(08)00607-8.

E. Rosch. Principles of Categorization. In E. Rosch and B. B. Lloyd, editors, *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum Associates, Hillsdale (NJ), USA, 1978. Reprinted in *Readings in Cognitive Science. A Perspective from Psychology and Artificial Intelligence*, A. Collins and E.E. Smith, editors, Morgan Kaufmann Publishers, Los Altos (CA), USA, 1991.

M. Sahimi. Flow phenomena in rocks: from continuum models to fractals, percolation, cellular automata, and simulated annealing. *Rev. Mod. Phys.*, 65:1393–1534, Oct 1993. doi: 10.1103/RevModPhys.65.1393.

L. M. Sander. Diffusion-limited aggregation: A kinetic critical phenomenon? *Contemporary Physics*, 41(4):203–218, 2000. doi: 10.1080/001075100409698.

P. Sarkar. A brief history of cellular automata. *ACM Comput. Surv.*, 32(1):80–107, March 2000. ISSN 0360-0300. doi: 10.1145/349194.349202.

M. Scheidgen. *Description of Languages Based on Object-Oriented Meta-Modelling*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät II, Humboldt-Universität zu Berlin, 2009.

M. Schlick. *Allgemeine Erkenntnislehre*, volume Moritz Schlick. Gesamtausgabe. Springer-Verlag Wien, 2009.

D. C. Schmidt. Model-driven Engineering. *Computer*, pages 25–31, Feb 2006.

J. Schnakenberg. Simple chemical reaction systems with limit cycle behaviour. *Journal of Theoretical Biology*, 81(3):389 – 400, 1979. ISSN 0022-5193. doi: 10.1016/0022-5193(79)90042-0.

L. Schruben and E. Yücesan. Modeling paradigms for discrete event simulation. *Operations Research Letters*, 13(5):265 – 275, 1993. ISSN 0167-6377. doi: DOI:10.1016/0167-6377(93)90049-M.

R. W. Sebesta. *Concepts Of Programming Languages*. Pearson Education India, 2004. ISBN 9788131701140.

D. Shapere. Objectivity, rationality, and scientific change. *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, 1984:pp. 637–663, 1984. ISSN 02708647.

F. Shiginah. *Multi-Layer Cellular DEVS Formalism for Faster Model Development and Simulator Efficiency*. PhD thesis, THE UNIVERSITY OF ARIZONA, DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, 2006.

W. Silvert. Object-oriented ecosystem modelling. *Ecological Modelling*, 68(1-2):91 – 118, 1993. ISSN 0304-3800. doi: DOI:10.1016/0304-3800(93)90110-E. Theoretical Modelling Aspects.

M. Sonnenschein and U. Vogel. Hierarchical Asymmetric Cellular Automata for Multiple-scale Modelling of Ecological and Socio-economic Systems. In *Proceedings of the 16th Conference EnviroInfo 2002*. IGU/ISEP, 2002.

J. Sprinkle. Analysis of a metamodel to estimate complexity of using a domain-specific language. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 13:1–13:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0549-5. doi: 10.1145/2060329.2060359.

R. Tairas, M. Mernik, and J. Gray. Models in Software Engineering. chapter Using Ontologies in the Domain Analysis of Domain-Specific Languages, pages 332–342. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-01647-9. doi: 10.1007/978-3-642-01648-6_35.

The Eclipse Foundation. Homepage: ATL - a model transformation technology, August 2012. URL `http://www.eclipse.org/atl/`.

F. Theisselmann, D. Dransch, and J. Fischer. Model-driven Development of Environmental Modeling Languages: Language and Model Coupling. In Peter Fischer-Stabel, Horst Kremers, Alberto Susini, and Volker Wohlgemuth, editors, *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools, 23rd International Conference on Informatics for Environmental Protection*, 2009a.

F. Theisselmann, D. Dransch, and Haubrock S. Service-oriented architecture for environmental modelling - the case of a distributed dike breach information system. In R.S. Anderssen, Braddock R.D., and L.T.H. Newham, editors, *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, July 2009b.

F. Theisselmann, F. Kühnlenz, C. Krüger, J. Fischer, and T. Lakes. How to reuse and modify an existing land use change model? Exploring the benefits of language-centered tool support. In Klaus Greve and Armin B. Cremers, editors, *EnviroInfo 2010 Integration of Environmental Information in Europe, Proceedings of the 24th International Conference on Informatics for Environmental Protection*, pages 678 – 688. Shaker Verlag Aachen, 2010.

W. R. Tobler. A computer movie simulating urban growth in the detroit region. *Economic Geography*, 46:234–240, 1970. ISSN 00130095. URL `http://www.jstor.org/stable/143141`.

T. Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D: Nonlinear Phenomena*, 10(1–2): 117 – 127, 1984. ISSN 0167-2789. doi: 10.1016/0167-2789(84)90254-9.

T. Toffoli and N. Margolus. *Cellular Automata Machines — a New Environment for Modeling*. MIT Press, Cambridge, MA, 1986.

J.-P. Tolvanen. *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. PhD thesis, University of Jyväskylä, 1998.

P.M. Torrens. Cellular Automata. pages 1 – 4, 2009. doi: DOI:10.1016/B978-008044910-4.00411-9.

J. T Townsend. Chaos theory: A brief tutorial and discussion. In A. F. Healy, S. M. Kosslyn, and R. M. Shiffrin, editors, *From Learning Theory to Connectionist Theory: Essays in Honor of W. K. Estes*, volume 1. Hillsdale, NJ: Lawrence Erlbaum Associates, 1992.

D. L. Turcotte. Self-organized criticality. *Reports on Progress in Physics*, 62(10):1377, 1999.

F. v. Evert, D. Holzworth, R. Muetzelfeldt, A. Rizzoli, and F. Villa. Convergence in integrated modelling frameworks. In R.M. Argent A. Zerger, editor, *COSIT 2005*, MODSIM 2005 International Congress on Modelling and Simulation, pages 745 – 750, Elliottville, NY, USA, 2005. Modelling and Simulation Society of Australia and New Zealand.

H. Vangheluwe. *Multi-Formalism Modelling and Simulation*. PhD thesis, Gent University, 2001.

H. Vangheluwe, J. d. Lara, and P. J. Mosterman. An introduction to multi-paradigm modelling and simulation. In *Fernando Barros and Norbert Giambiasi, editors, Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems)*, pages 9–20, Lisboa, Portugal, April 2002.

G. Y. Vichniac. Simulating physics with cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):96 – 116, 1984. ISSN 0167-2789. doi: DOI:10.1016/0167-2789(84)90253-7.

F. Villa. Integrating modelling architecture: a declarative framework for multi-paradigm, multi-scale ecological modelling. *Ecological Modelling*, 137:23–42, June 2001.

F. Villa. A semantic framework and software design to enable the transparent integration, reorganization and discovery of natural systems knowledge. *J. Intell. Inf. Syst.*, 29(1): 79–96, 2007. ISSN 0925-9902. doi: http://dx.doi.org/10.1007/s10844-006-0032-x.

F. Villa, M. Donatelli, A. Rizzoli, P. Krause, S. Kralisch, and F. E. v. Evert. Declarative modelling for architecture independence and data/model integration: a case study. In Anthony J. Jakeman Alexey Voinov and Andrea Rizzoli, editors, *Proceedings of the iEMS Third Biennial Meeting: Summit on Environmental Modeling and Software*, July 2006.

F. Villa, I. N. Athanasiadis, and A. E. Rizzoli. Modelling with knowledge: A review of emerging semantic approaches to environmental modelling. *Environmental Modelling & Software*, 24(5):577 – 587, 2009. ISSN 1364-8152. doi: DOI:10.1016/j.envsoft.2008.09.009.

N. Villa-Vialaneix, M. Follador, M. Ratto, and A. Leip. A comparison of eight metamodeling techniques for the simulation of N2O fluxes and N leaching from corn crops. *Environmental Modelling & Software*, 34(0):51 – 66, 2012. ISSN 1364-8152. doi: 10.1016/j.envsoft.2011.05.003. <ce:title>Emulation techniques for the reduction and sensitivity analysis of complex environmental models</ce:title>.

A. Voinov, C. Fitz, R. Boumans, and R. Costanza. Modular ecosystem modeling. *Environmental Modelling & Software*, 19:285–304, Mar 2004.

H.J.M. (Bert) de Vries. Environmental modelling. In Jan J. Boersema and Lucas Reijnders, editors, *Principles of Environmental Sciences*, chapter 17, pages 345 – 373. Springer Science + Business B.V., 2009.

G. Wainer. Cell-DEVS Modeling of Environmental Applications. In Alexey Voinov, Anthony J. Jakeman, and Andrea E. Rizzoli, editors, *Proceedings of the iEMSs Third Biennial Meeting: "Summit on Environmental Modelling and Software"*. International Environmental Modelling and Software Society, Burlington, USA, July 2006.

W. E. Walker, P. Harremoës, J. Rotmans, J. P. Van Der Sluijs, M. B. A. Van Asselt, P. Janssen, and M. P. K. Von Krauss. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. *Integrated Assessment*, 4(1): 5–17, 2003.

T. Walter, F. Silva Parreiras, and S. Staab. OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In Andy Schürr and Bran Selic, editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 408–422. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04424-3.

Fred G. R. Watson and Joel M. Rahman. Tarsier: a practical software framework for model development, testing and deployment. *Environmental Modelling & Software*, 19:245–260, Mar 2004. URL `http://www.sciencedirect.com/science/article/B6VHC-49KH3CS-1/2/12d68c38f66f7defdcc2d7d34fe7b7c8`.

D. A. Watt. *Programming Language Design Concepts*. John Wiley & Sons, 2004. ISBN 0470853204.

I. D. White, D. Mottershead, and S. J. Harrison. *Environmental systems : an introductory text*. G. Allen & Unwin, London ; Boston :, 1984. ISBN 0045510652 0045510644.

D. Wile. Lessons Learned from Real DSL Experiments. *Hawaii International Conference on System Sciences*, 9:325b, 2003. doi: http://doi.ieeecomputersociety.org/10.1109/HICSS.2003.1174893.

E. Winsberg. Sanctioning Models: The Epistemology of Simulation. *Science in Context*, 12(02):275–292, 1999. doi: 10.1017/S0269889700003422.

S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, (96):15–57, 1984.

S. Wolfram. *A New Kind of Science*. Wolfram Media, January 2002.

T. Worsch. Simulation of cellular automata. *Future Generation Computer Systems*, 16 (2-3):157 – 170, 1999. ISSN 0167-739X. doi: DOI:10.1016/S0167-739X(99)00044-8.

T. Worsch. Cellular Automata as Models of Parallel Computation. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 741–755. Springer, 2009. ISBN 978-0-387-75888-6.

J. Wu and J. L. David. A spatially explicit hierarchical approach to modeling complex ecological systems: theory and applications. *Ecological Modelling*, 153(1–2):7 – 26, 2002. ISSN 0304-3800. doi: 10.1016/S0304-3800(01)00499-9.

Y. Wu, F. Hernandez, F. Ortega, P. J. Clarke, and R. France. Measuring the effort for creating and using domain-specific models. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 14:1–14:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0549-5. doi: 10.1145/2060329.2060360.

xText. *Xtext 2.3 Documentation*, June 2012. URL `http://www.eclipse.org/Xtext/documentation/2.3.0/Documentation.pdf`.

B. P. Zeigler. System-Theoretic Representation of Simulation Models. *IIE Transactions*, 16(1):19 – 34, 1984.

B. P. Zeigler and H. S. Sarjoughian. *Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models*, 2005.

B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, San Diego, 2nd edition, 2000.

H. Zemanek. Semiotics and programming languages. *Commun. ACM*, 9(3):139–143, March 1966. ISSN 0001-0782. doi: 10.1145/365230.365249.

Y. Zhao, S. A. Billings, and A. F. Routh. IDENTIFICATION OF THE BELOUSOV-ZHABOTINSKII REACTION USING CELLULAR AUTOMATA MODELS. *International Journal of Bifurcation and Chaos*, 17(05):1687–1701, 2007. doi: 10.1142/S0218127407017999.

List of Figures

Listings

List of Algorithms

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Dissertationsschrift selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Weiterhin erkläre ich, dass ich mich nicht anderwärts um einen Doktorgrad beworben habe und keinen Doktorgrad im Promotionsfach Informatik besitze.

Ich habe Kenntnis der Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin gemäß des Amtlichen Mitteilungsblattes Nr. 34/2006.

Berlin, den 9. Januar 2013 Falko Martin Theisselmann